НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

«До захисту допущено»
В.о. завідувача кафедри
_____ __*О.А.Павлов*__
(підпис) (ініціали, прізвище)
“___” _____ 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки *6.050103 «Програмна інженерія»*

спеціальність *«Програмне забезпечення систем»*

на тему: *«Програмні засоби для розподіленого зберігання даних»*

Виконав: студент 4 курсу, групи ІП-51

Ядельський Ростислав Ігорович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *асист. Ісаченко Г.В.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант з графічної документації *доц. к.т.н. Ліщук К.І.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент *ст. викл. каф. ОТ Виноградов Ю.М.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

АНОТАЦІЯ

Пояснювальна записка дипломного проекту складається з чотирьох розділів, містить 9 рисунків, 39 таблиць, 11 джерел – загалом 71 сторінка.

Об`єкт дослідження: програмні засоби для розподіленого зберігання даних.

Мета дипломного проекту: покращення методів зберігання оперативних даних у розподіленому сховищі даних.

У першому розділі виконано аналіз предметної області, відомих технічних рішень, сформульовано функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення.

У другому розділі було виконано моделювання програмного забезпечення. Побудовано схему бізнес-процесів та описано принцип роботи сховища. Також було спроектовано архітектуру програмного забезпечення, розроблено API його та обґрунтовано вибір технологій для його реалізації.

У третьому розділі описано специфіку тестування та налагодження програмного забезпечення, описано процеси тестування та наведено контрольний приклад.

У четвертому розділі описано впровадження програмного забезпечення.

Також наведено: опис програми, технічне завдання, програму та методику тестування, керівництво користувача, керівництво адміністратора та графічний матеріал.

КЛЮЧОВІ СЛОВА: СХОВИЩА ДАНИХ, РОЗПОДІЛЕНІ СИСТЕМИ, ЗБЕРІГАННЯ ДАНИХ, КЛАСТЕР.

ABSTRACT

The explanatory note of the diploma project consists of 9 pictures, 39 tables and 11 sources – total of 71 pages.

The object of study: software for distributed dat storing.

The aim of the diploma project: to improve methods for storing operational data in distributed data storage.

In the first section the subject area was analyzed, existing technical solutions were investigated, functional and non-functional requirements for the developed software were formulated.

In the second section software modeling was performed. Also a business program diagram was constructed and storing methods were described. The software architecture was designed, its API was developed and the choice of technologies for the realization was grounded.

The third section describes the specifics of software testing and debugging, describes the testing processes and gives a checklist.

The fourth section describes the application deploying and maintenance.

This work also includes following materials: a description of the program, a specification, a program and test methods, a user's guide, and graphic materials.

KEYWORDS: DATA STORAGES, DISTRIBUTED SYSTEMS, DATA STORING, CLUSTER.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ВСТУП

Вимоги до сучасного програмного забезпечення сильно змінились за останні десятиліття. На сьогоднішній день набули високого розповсюдження сервіси, якими можуть одночасно користуватись мільйони людей, простій в їх роботі всього на кілька секунд може привести до надзвичайно великих збитків. Для функціонування таких сервісів часто необхідні спеціальні тими сховищ – розподілені сховища даних.

Вони являють собою сукупність логічно зв’язаних даних, розподілених у комп’ютерній мережі. Система управління розподіленим сховищем даних забезпечує прозорий доступ до даних. Завдяки тому, що дані одночасно зберігаються на кількох вузлах системи, можна забезпечити високу надійність такого сховища, так як при виході з ладу кількох вузлів система в цілому продовжить працювати. Також це дозволяє зберігати об’єми даних, що фізично не можуть бути оброблені на одному вузлі та пришвидшити доступ до них. Крім цього кількість вузлів можна динамічно змінювати в залежності від навантаження на систему, що є великою перевагою.

Попри всі описані вище переваги, при використанні розподілених сховищ даних виникає чимало обмежень і труднощів пов’язаних з особливістю їх реалізації. Так звана CAP теорема говорить, що у розподіленій системі одночасно можна досягнути лише двох з трьох перелічених властивостей:

– узгодженість даних (у будь-який момент часу усі вузли системи оперують однаковими даними);

– доступність (кожен запит отримає відповідь у прийнятний час);

– стійкість до розділення (попри розділення на ізольовані секції або втрати зв'язку з частиною вузлів, система не втрачає стабільність і здатність коректно відповідати на запити).

Тому на сьогоднішній день існує достатньо велика кількість абсолютно різних за своїм принципом роботи розподілених сховищ, що застосовують різні підходи до забезпечення оптимального співвідношення між цими трьома

властивостями, однак при використанні будь-якого з наявних рішень доведеться миритись з багатьма компромісами. Такі системи часто дозволяють обирати оптимальні для конкретного випадку параметри сховища при його початковій інсталяції, після чого конфігурації не змінюються. Однак в сучасних реаліях такий підхід є не достатньо гнучким, адже часто необхідно мати можливість змінювати параметри системи такі як кількість вузлів чи рівень реплікації даних динамічно, під час роботи самого сховища. Також великим недоліком багатьох сучасних сховищ даних є їх підхід до розв'язування конфліктів при одночасній зміні даних на кількох вузлах системи, адже переважна більшість розподілених сховищ віддають перевагу останній за часовою міткою модифікації, що далеко не завжди є правильним і може приводити до втрати даних.

Саме тому мета даної роботи – розробити розподілене сховище даних, що дозволить динамічно змінювати свої параметри, такі як кількість вузлів у системі, рівень реплікації даних та алгоритм розв'язання конфліктів. Така гнучкість дозволить оптимізувати систему для конкретних сценаріїв використання, та не буде накладати багатьох обмежень розподілених систем на користувачів сховища.

Завданням даної роботи є програмна реалізація даного сховища, що свідчить про дієздатність розробленої системи, та можливість його практично застосування. Компонентами розроблюваної системи є серверний застосунок з описаним API та панеллю з метриками, що показують актуальну інформацію по роботі системи. Результатом роботи є розподілене сховище даних, яке можна застосовувати як елемент будь-якої системи, що потребує збереження та обробки свого стану.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Розподілене сховище даних – це сховище даних, в якому інформація зберігається на одному чи багатьох фізичних вузлах, що забезпечує прозорий доступ до даних, приховуючи від користувача їх реальне розміщення. [6]

У таких сховищ є ряд можливостей, що суттєво відрізняють їх від нерозподілених аналогів, а саме відмовостійкість, здатність до горизонтального масштабування та краща швидкодія при великому навантаженні.

Розподілені сховища даних зазвичай виконують такі методи, як фрагментація та реплікація даних при своїх роботі.

Фрагментація даних – підхід, при якому логічно пов’язаний набір даних розділяють на фрагменти з метою їх зберігання кожного фрагмента на окремому вузлі системи. Цей підхід дозволяє зменшити навантаження на окремі вузли та змінювати кількість вузлів на яких зберігається інформація в залежності від її кількості. Також при виході з ладу кількох вузлів, інформація, що знаходиться на інших вузлах все ще залишається доступною. [7] Є три типи фрагментації даних: горизонтальна, вертикальна та змішана. Різницю між ними можна пояснити на прикладі реляційних баз даних.

При горизонтальній фрагментації відбувається розділ таблиці на групи кортежів тобто рядків таблиці за вказаним методом. Таку фрагментацію зазвичай називають шардінгом. Перевагами цього підходу є можливість паралельно обробляти кортежі, а також розділити їх таким чином, щоб дані знаходились там, де вони будуть використовуватись найчастіше.

При вертикальній фрагментації дані розділяються не по рядкам, а по стовбцям таблиці. Таким чином, необхідно щоб з кожним елементом кортежу також записувався асоційований з ним ідентифікатором кортежу. Цей підхід дозволяє поділити частини кортежів так, щоб вони знаходились так, де їх

використовують найчастіше, а також виконувати з’єднання фрагментів ефективно.

При використанні змішаної фрагментації одночасно застосовуються як вертикальна, так і горизонтальна фрагментація.

Реплікація даних – це метод, що передбачає зберігання копії даних на різних вузлах для збільшення стійкості системи до відмов та для пришвидшення доступу до даних. Дані вважаються реплікованими, якщо їх копії знаходяться на двох та більше вузлах, а самі копії називаються репліками. Кількість копій даних називається рівнем реплікації. При повній реплікації дані копіюються на всі вузли системи. [7]

Реплікація надає наступні переваги:

– висока доступність даних, при виході з ладу вузла, дані, що зберігаються на ньому все ще є доступними на інших репліках;

– паралелізм зчитування даних, так як дані можна зчитувати з будь-якої репліки одночасно;

– зниження вартості передачі даних, адже при читанні можна обрати найближчу репліку.

Попри описані вище переваги існують і певні недоліки, а саме:

– пропорційно до рівня реплікації підвищується вартість зберігання даних, так як необхідно зберігати і їх копії;

– дуже сильно ускладнюється підтримка цілісності даних, адже будь-які зміни необхідно записувати на всі репліки, а це відбувається не миттєво.

Таким чином при використанні реплікації виникає проблема підтримки узгодженості даних. Є чимало підходів до її вирішення, однак на практиці зазвичай використовують або механізм двох крокової фіксації, для того, щоб впевнитись у правильності всіх даних, або послаблюють вимоги до системи, жертвуючи узгодженістю даних на користь швидкодії, використовуючи так звану узгодженість даних з часом, яка означає, що дані можуть бути

неконсистентні протягом певного періоду часу після їх модифікації, однак з часом система прийде до узгодженого стану.

Серед розподілених сховищ даних досить розповсюдженими є, так звані, “ключ-значення” сховища – це тип сховищ даних, що працюють за принципом асоціативного масиву. Такі сховища зазвичай підтримують всього три базові операції: збереження, отримання та видалення даних за вказаним ключем. Такий підхід до зберігання інформації сильно відрізняється від реляційних баз даних, однак він є широко розповсюджений, так як дозволяє дуже просто проводити фрагментацію, реплікацію, такі сховища мають хорошу швидкодію.

1.2 Змістовний опис і аналіз предметної області

1.2.1 CAP теорема

Основна проблема усіх розподілених систем виражена у CAP теоремі, назва якої є акронімом термінів consistency, availability та partition tolerance. Вона говорить, що будь-яка розподілена система не може одночасно забезпечити виконання більше двох із трьох властивостей:

– узгодженість даних, тобто гарантія, що у будь-який момент часу усі вузли системи оперують однаковими даними;

– доступність – гарантія отримання відповіді на будь-який запит у прийнятний час;

– стійкість до розділення – властивість за наявності попри розділення на ізольовані секції або втрати зв'язку з частиною вузлів, система не втрачає працездатності і здатності коректно відповідати на запити. [8]

Таким чином всі розподілені системи можна розділити на три класи.

Клас CA передбачає систему в якій наявна доступність та узгодженість даних, однак не може забезпечуватись стійкість до розділення. До таких систем відносяться програми, що гарантують виконання ACID вимог, наприклад реляційні бази даних.

Клас AP включає в себе розподілені системи, в яких не гарантується цілісність даних, однак вони підтримують високу доступність, а також є стійкими до розділення при виникненні неполадок мережі. Прикладом AP системи є DNS, вони є достатньо популярними серед нереляційних баз даних, адже достатньо часто узгодженість даних не є настільки важливою, як два інші параметри CAP теореми.

До класу CP належать системи, що забезпечують цілісність даних на всіх вузлах і здатність працювати при розділенні мережі, однак вони можуть не відповідати на певну частину запитів. Велика частина фінансових систем входять до цього класу, адже у них узгодженість даних має найвищий пріоритет, найпростішим прикладом є мережа банкоматів.

Варто зауважити, що на практиці ці властивості зазвичай розглядають не як бінарні признаки, а як якісні, тобто характеризують систему певним рівнем узгодженості, доступності та стійкості до розділення в залежності від умов використання. Слід також зазначити, що переважну більшість часу розподілені системи перебувають у нормальному стані, при якому немає мережевих неполадок між вузлами.

1.2.2 Фрагментація даних

Сховища даних “ключ-значення” у загальному випадку підтримують тільки горизонтальну фрагментацію. Є кілька підходів до її реалізації, однак вони всі визначаються з допомогою наступної функції $f: K \to N$, де K – це множина ключів даних, а N – множина номерів вузлів системи, що пронумеровані послідовно від 0 до C не включно, де C рівне кількості вузлів системи. Назвемо її функцією фрагментації. Для реалізації такої функції можна використати заздалегідь визначену хеш-функцію $h: K \to \mathbb{Z}$. Використавши її отримаємо наступну формулу:

$$f(k) = h(k) \bmod C;\ k \in K$$

Приклад застосування цього метода для системи з 3 вузлів вказано у таблиці 1.1.

Таблиця 1.1 – Результат роботи простого хешування

Ключ	Хеш	Номер вузла
“foo”	1 633 428 562	2
“bar”	7 594 634 739	0
“world”	5 000 799 124	1
“count”	9 787 173 343	0

Якщо хеш-функція *h* є хорошою, тобто її результатом буде величина, яку складно відрізнити від випадкової, то данні будуть рівномірно розподілені на всіх вузлах, однак у такого підходу є один недолік. Так як, розподілених системах час від часу кількість вузлів може змінюватись, як через зміну навантаження, так і через вихід з ладу вузлів, критичною є кількість даних, які необхідно перерозподілити по вузлах при зміні кількості вузлів. Так при використанні описаного вище метода, додавання чи вилучення вузла приведе до перерозподілу практично всіх даних системи, що є неприпустимим.

Цю проблемо вирішують з допомогою узгодженого хешування. Його принцип роботи наступний: всю числову вісь, значень якої можуть набувати хеші ключів, “закручують” у кільце, таким чином, щоб її початок співпадав з кінцем. Після цього кожному вузлу системи присвоюють випадкове значення на цій осі, назвемо його міткою вузла. Тепер для знаходження вузла, на якому зберігається значення асоційоване з вказаним ключем знаходять хеш ключа, після чого обирають вузол найближчий до значення хешу в напрямку зростання значень числової осі. [1]

Нехай хеш-функція може приймати значення від 0 до 2^{32} та в системі є три вузла. Кожному з вузлів присвоєно випадкове значення з цього інтервалу:

Таблиця 1.2 – Мітки вузлів

Вузол	Мітка
0	1 431 656 765
1	3 292 808 260
2	4 151 801 719

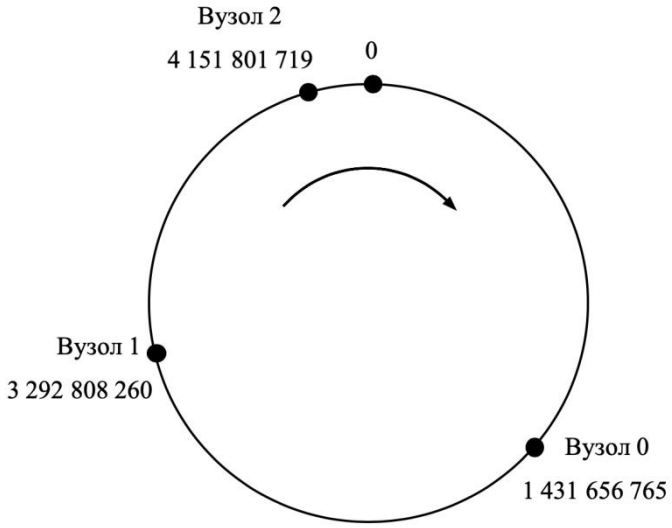

Рисунок 1.1 – Візуальне представлення узгодженого хешування

Візуально це показано на рисунку 1.1.

У таблиці 1.3 вказаний результат роботи даного алгоритму.

Таблиця 1.3 – Результат роботи узгодженого хешування

Ключ	Хеш	Номер вузла
“foo”	2 343 423	0
“bar”	3 839 939 111	2
“world”	1 500 323 432	1
“count”	4 188 323 434	0

При додаванні вузла, йому присвоюється випадкова мітка. Таким чином на доданий вузол слід перемістити лише ті дані, значення хешів яких потрапляють в інтервал між міткою вузла, що знаходиться до доданого та міткою доданого вузла. Для видалення вузла, ми просто видаляємо його мітку,

а дані, що зберігались на ньому переміщаємо на вузол, мітка якого є наступною. Таким чином додавання, чи видалення вузла потребує переміщення даних всього з одного вузла системи, тобто в середньому слід перемістити всього $\frac{K}{C}$ ключів, де K – це загальна кількість ключів в системі. Однак у випадках, коли кількість вузлів невелика, даний через випадкове призначення міток, на деякі вузли може припадати набагато більше даних, ніж на інші. Щоб вирішити цю проблему використовують так звані віртуальні вузли. Для цього кожен вузол асоціюють з M віртуальними вузлами, і вже цим віртуальним вузлам присвоюють мітки, при тому значення M зазвичай обирають більшим ніж 10, таким чином суттєво зменшуючи ймовірність нерівномірного розподілу ключів.

Підсумовуючи все вище сказане, можна оцінити асимптотичну складність алгоритмів фрагментації на основі простого хешування та узгодженого хешування. Результати наведені в таблиці 1.4.

Таблиця 1.4 – Асимптотична складність операцій при різних типах хешування

Тип дії	Просте хешування	Узгоджене хешування
Додавання вузла	$O(K)$	$O(\frac{K}{C} + \log(C))$
Видалення вузла	$O(K)$	$O(\frac{K}{C} + \log(C))$
Доступ до елементу	$O(1)$	$O(\log(C))$

Член $O(\log(C))$ відповідає за пошук мітки відповідного вузла, що реалізується з допомогою бінарного пошуку по відсортованій за значенням мітки таблиці з вузлами. [2]

1.2.3 Реплікація даних

Як вже було сказано, реплікацію застосовують для забезпечення доступності даних та відмовостійкості сховища. Реплікація за часом відпрацювання поділяється на синхронну та асинхронну та за моделлю роботи на реплікацію даних та реплікацію транзакцій.

Реплікація є синхронно, коли будь-яка зміна в даних не вважається завершеною поки вона не буде записана на всі репліки. Зазвичай для реалізації такої поведінки використовують механізм двох крокової фіксації.

При асинхронній реплікації відповідь на запит зміни даних відбувається не очікуючи на процес завершення копіювання на всі репліки, тобто реплікація працює у фоні.

Реплікація даних передбачає, що при модифікації даних на репліки буть скопійовані вже модифіковані дані, що повністю замінять свої прообрази.

При реплікації транзакцій, копіюються не самі данні, а опис їх модифікацій і за рахунок того, що на всіх репліках історія модифікацій даних буде однаковою, результуючі дані теж співпадатимуть.

1.2.4 Обмін інформацією у розподіленій системі

Обмін інформацією та управління єдиним станом в розподіленій системі достатньо складна задача, адже для цього необхідно розіслати інформацію всім членам кластеру, враховуючи можливість мережевих збоїв, не повної зв’язності мережі, а також динамічного додавання та видалення вузлів кластера. Для вирішення цієї задачі використовують так званий gossip протокол.

Для того щоб пояснити принцип його роботи можна провести аналогію з офісними працівниками, що поширюють плітки. Кожної години вони збираються на кухні та кожен працівник ділиться плітками з іншими випадково вибраними колегами. Спочатку працівник А розповідає нову плітку працівникам Б. За працівник А розповідає колезі В цю чутку, а працівник Б розповідає про неї Г. Таким чином кількість ознайомлених з чуткою зростає

вдвічі щогодини, якщо не враховувати дублювання в межах одного сеансу. [3] Так повторюється поки всі працівники не будуть ознайомлені. Демонстрацію роботи цього алгоритму показано на рисунку 1.2.

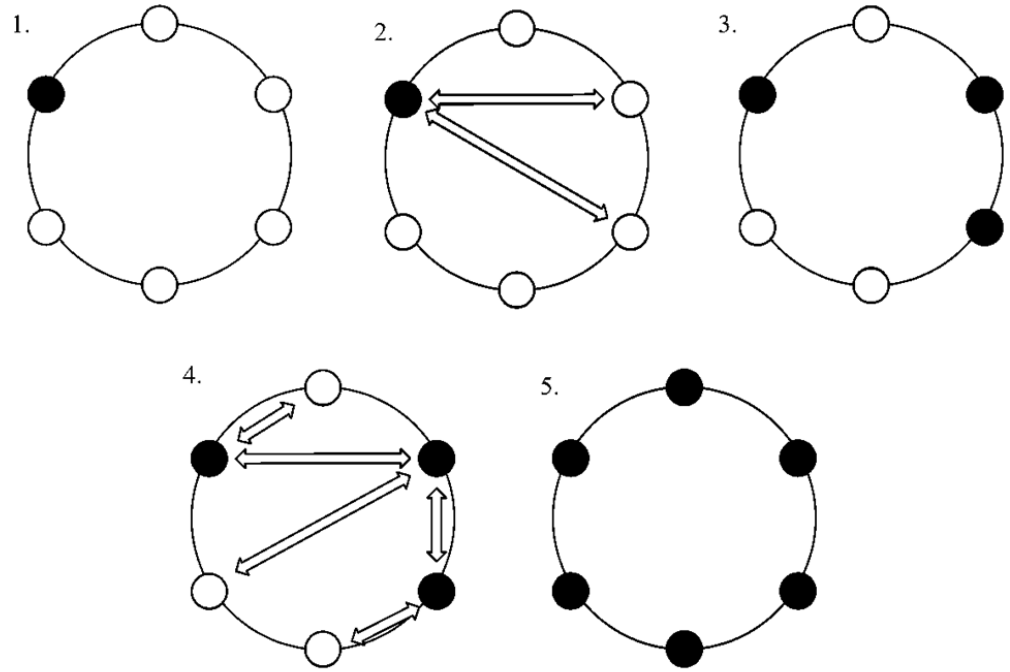

Рисунок 1.2 – Робота gossip протоколу

Даний протокол дозволяє сигналізувати всі вузли системи про зміни в її конфігурації, такі як появу нового вузла чи відключення одного з вузлів.

1.2.5 Вирішення конфліктів у даних

У розподіленій системі, кілька вузлів якої можуть обробляти модифікацію одних і тих же даних без можуть виникати конфлікти в даних. Для вирішення таких ситуацій існує кілька підходів.

Найпопулярніший та найпростіший з них – зберігати час модифікації даних, та віддавати перевагу даним, що були модифіковані останніми. Однак у такого підходу є суттєвий недоліки. По перше, системні годинники на різних вузлах неможливо ідеально синхронізувати, в наслідок чого можливі неправильні результати. По друге, реплікація даних не відбувається миттєво, тому навіть якщо системні годинники повністю співпадають, можливі ситуації,

коли данні просто не встигли копіюватись і тоді прообразом модифікованих даних буде їх застаріла версія.

Набагато кращим з точки зору узгодженості даних є підхід з використанням так званих вільних від конфліктів прireплікації типів даних (CRDTs). Це спеціальні структури даних, що при реплікації у комп'ютерній мережі, та одночасній модифікації на різних репліках без їх координації, можуть бути об'єднані в єдину структуру без втрат даних, тобто для всіх конфліктних ситуацій в них є коректне математичне рішення. [5; 11] Такі структури широко розповсюджені в розподілених обчисленнях, однак вони достатньо обмежені за своїми можливостями, тому їх зазвичай не вистачає для вираження всієї необхідної бізнес модемі.

Рішенням цієї проблеми може бути підхід, що суміщає два попередні. Для цього можна використати так званий векторний годинник, що є вільною від конфліктів при реплікації структурою даних, яка дозволяє однозначно визначати конфліктні ситуації між двома наборами даних. Ця структура складається з вектора, розмірність якого співпадає з кількістю реплік. Зі всіма записами у сховищі зберігається їх векторний годинник. При модифікації запису на вузлі, елемент вектора що асоційований з даним вузлом збільшується на одиницю. Коли відбувається реплікація, векторні годинники конфліктуючих даних поелементно порівнюються, і якщо кожен елемент одного вектора більший за відповідний елемента іншого вектора, результатом розв'язання конфлікту будуть данні асоційовані з більшим вектором. Якщо ж це не так, то відбулась одночасна модифікація даних на двох репліках, тому при спробі зчитати дані для їх подальшої модифікації конфліктуючі версії даних будуть надіслані клієнту, який і виконає розв'язання конфлікту. [1; 4]

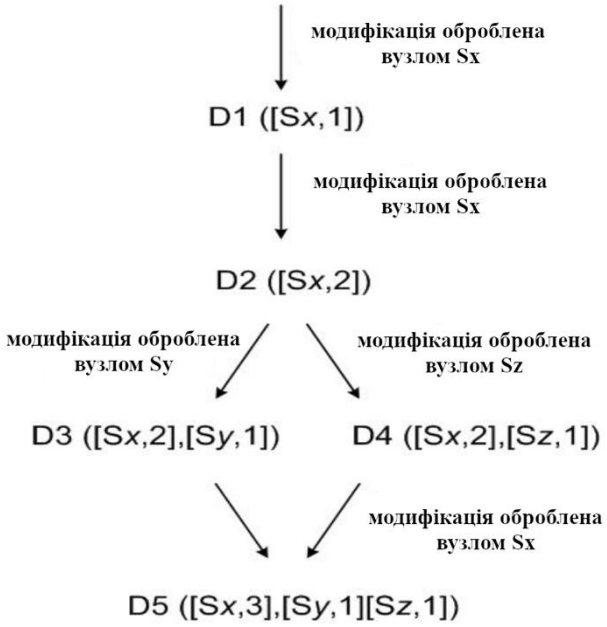

Рисунок 1.3 – Робота векторних годинників

На рисунку 1.3 зображено приклад роботи даного алгоритму. Спершу на вузол Sx поступає запит на збереження нового об’єкту, що приводить систему в стан D1. Після цього на вузол Sx приходить запит на модифікацію об’єкту, в наслідок чого значення відповідного елемента векторного годинники збільшується на одиницю. Після цього на репліки Sy та Sz одночасно приходять запити на модифікацію даних, в наслідок чого утворюється дві конфліктні версії D3 та D4. Тепер при спробі модифікації даних, користувач що спробує їх зчитати, отримає у відповідь ці дві версії, вирішить конфлікт та надішле отриманий результат на репліку Sx, в наслідок чого у системі вирішаться всі конфлікти.

1.3 Аналіз успішних IT-проектів

1.3.1 Аналіз відомих технічних рішень

На сьогоднішні існує велика кількість розподілених сховищ даних, як вже було сказано, за CAP теоремою їх можна поділити на 3 типи.

Сховища, що підтримують узгодженість даних та їх доступність мають наступні особливості:

– підтримка синхронної та асинхронної реплікації;

– система транзакцій з використанням двох крокової фіксації для підтримки узгодженості даних;

– при виникненні розділу мережі втрачається можливість взаємодії з системою;

– зазвичай у таких сховищ реплікація працює за принципом мастер-слейв;

– крім інструментів для збереження даних доступні засоби для їх аналізу та обробки;

– в переважній більшості представляють дані з бопомогою реляційної моделі.

Системи, що продовжують працювати при розділенні мережі, однак при цьому втрачають працездатність деяких вузлів характеризуються такими властивостями:

– реплікація, як і в попередньому випадку працює за принципом мастер-слейв;

– при відділенні мастера відбувається його автоматична заміна;

– у випадку розділу мережі система перестане обробляти модифікації даних поки не буде мати гарантії їх безпечного завершення;

– як і минулий клас, містять засоби аналізу і обробки даних.

Сховища підтримуючі високу доступність та стійкі до розділу мережі відрізняються наступним:

– відсутні транзакції;

– конфлікти у даних вирішуються на користь останньої модифікації;

– кожен вузол системи може обробляти модифікації даних;

– підтримують узгодженість даних з часом.

1.3.2 Аналіз відомих програмних продуктів

Найпопулярніші розподілені сховища даних:

Cassandra

Розподілена нереляційна система управління базами даних, що оперує так званими сімействами стовбців, тобто представляє дані як таблицю, однак у своїй реалізації схожа на сховища “ключ-значення”. Розроблена для забезпечення високої швидкодії, та надійності в умовах надзвичайно великого навантаження, та може зберігати надвеликі об’єми даних.

Для зручності доступу для даних використовує мову формування структурованих запитів CQL (Cassandra Query Language), що нагадує урізаний по функціоналу SQL, в якому підтримуються лише прості запити з фільтрацією та модифікація даних. Кожен її вузол може обробляти модифікацію даний, та поширювати її з допомогою асинхронної реплікації. Таким чином дана система продовжує працювати навіть в умовах розділення мережі.

У своїй імплементації базується на узгодженому хешуванні, що забезпечує можливість до практично лінійного горизонтального масштабування, яке можна проводити без простою. Може працювати на кластерах, що містять сотні та навіть тисячі вузлів та зберігати данні об’ємом понад 1Пб. Підтримує узгодженість даних з часом, відповідно не підтримує створення транзакції, а при вирішенні конфліктів віддає перевагу останньому за часовою міткою запису.

Cassandra знаходиться у вільному розповсюдженні, а її сирцевий код відкритий.

DynamoDB

Розподілене “ключ-значення” сховище даних, що забезпечує час відповіді в межах 10 мілісекунд не залежно від навантаження та об’єму даних, що зберігається. Такого результату досягають завдяки тому, що DynamoDB функціонує як сервіс у складі Amazon Web Services, тобто дана система має доступ до практично необмежених обчислювальних потужностей та запасів пам’яті.

Як і Cassandra базується на консистентному хешуванні, а всі її вузли є рівнозначними. Підтримує дві моделі узгодженості даних: зчитування потенційно неконфліктних даних, що дозволяє досягти максимальної пропускної здатності, однак може не відображати останніх модифікацій у даних, так як реплікація відбувається в межах секунди, та зчитування узгоджених даних, що гарантує отримання останньої версії даних жертвуючи швидкодією.

Підтримує тільки операції вставки, видалення, оновлення та зчитування даних за вказаним ключем.

DynamoDB може обробляти більше ніж 10 трильйонів запитів на день при піковому навантаженні більше 20 мільйонів запитів на секунду.

MongoDB

Документно-орієнтована система керування базами даних, що оперує даними у JSON-подібному форматі та не потребує вказування схеми даних. Доступ до даних забезпечується досить гнучкою мовою для написання запитів, які виконуються за принципом MapReduce.

Всі запити виконуються основним вузлом системи, та реплікуються на інші вузли. Якщо головний вузол перестає відповідати, автоматично обирається новий головний вузол та запити перенаправляються на нього. Таким чином, система хоч і є централізованою, роль управляючого вузла може на себе взяти будь-яка репліка. Однак сам вузол, що відділився не може обробляти запити, тому дане сховище не є стійким до розділу мережі.

Підтримує кілька типів шардінгу, а саме за діапазонами значень ключів, на основі простого хешування ключів, та з ручним заданням співвідношення між діапазоном значень ключів та відповідним фрагментом.

MongoDB забезпечує узгодженість даних, та в останніх версіях з’явилась підтримка транзакцій, однак їх широке використання не рекомендується, так як це суттєво зменшує швидкодію системи.

Як і Cassandra, сирцевий код даної СУБД є відкритим та вона є вільною для розповсюдження.

Etcd

Сховище даних “ключ-значення”, розроблене для збереження критичних даних з фокусом на швидкодію, надійність, простоту та безпеку. Etcd не підходить для збереження великих об’ємів даних, так як обмін інформацією в ньому відбувається з допомогою Raft протоколу узгодження. В переважній більшості використовують для збереження службової інформації, що необхідна для функціонування розподіленої системи.

Etcd може опрацьовувати до 10 тисяч модифікацій даних в секунду. Його розвертають на кластерах величиною до кількох десятків вузлів, так як через використання Raft протоколу при збільшенні кількості вузлів, дуже сильно зростає навантаження на мережу.

У даній системі підтримується узгодженість даних з часом. Відповідно транзакцій немає. Так як це сховище “ключ-значення”, Etcd підтримує тільки базові операції читання, запису. видалення та модифікації даних за ключем.

Bigtable

Розподілене сховище даних, призначене для зберігання надвеликих об’ємів інформації, що оперує родинами стовбців, що містять єдиний ключ, що можна уявляти як сховище даних “ключ-значення”, значеннями якого є асоціативні масиви. Bigtable є комерційною розробкою Google, та працює як сервіс в межах Google Cloud Platform, таким чином, єдиний спосіб його використання – інтеграція з GCP.

Дане сховище може зберігати понад 1Пб інформації, та забезпечувати доступ до неї із затримкою, меншою за 10 мілісекунд. Воно підтримує практично не обмежене горизонтальне масштабування, при чому без зупинки системи.

Для доступу до даних використовуються запити на основі інтервалів у просторі ключів. Так як дані збережені у лексикографічному порядку ключів,

так запити виконуються дуже швидко, але потребують щоб вся необхідна для запиту інформація знаходилась в ключі.

Bitable підтримує до 4 кластерів для реплікації. За замовчуванням в забезпечується узгодженість даних з часом, однак при певних налаштуваннях реплікації можна досягти як узгодженості при читання власних модифікацій, так і повної узгодженості даних, однак це зробить систему повільнішою та менш надійною.

Розглянувши всі вище перелічені системи можна виділити їх основні переваги та недоліки. Частина з низ підтримують динамічне горизонтальне масштабування та фрагментацію. Всі вони реалізують реплікацію даних. Однак деякі з сховищ не можуть працювати в умовах розділу мережі. У більшості розглянутих систем забезпечується узгодженість даних з часом, що дає їм можливість працювати під високим навантаженням та бути стійкими до відмов вузлів, проте при розв’язанні конфліктів вони полягаються тільки на часову мітку. До того ж не у всіх з них є можливість динамічної зміни рівня узгодженості.

1.4 Аналіз вимог до програмного забезпечення

Для того щоб визначити вимоги до програмного забезпечення розглянемо наступні типи його користувачів:

- системний адміністратор;
- моніторинг спеціаліст;
- програміст;
- клієнтський додаток.

Системний адміністратор налаштовує мережу між вузлами, інсталює систему та запускає її.

Моніторинг спеціаліст спостерігає за станом системи та повідомляє системного адміністратора чи програміста у випадку виявлення аномалії в її

роботі. Для цього вся необхідна інформація відображається на відповідній веб сторінці.

Програміст розробляє клієнтський додаток, що який працюватиме з сховищем використовуючи його публічне API.

Системний адміністратор за необхідності може додати чи вилучити вузол з мережі.

Загалом система реалізується як серверний застосунок та повинна мати наступні можливості:

- запуск на одному та кількох вузлах;
- приєднання нових вузлів з подальшим перерозподілом даних по них;
- вилучення вузлів з системи з подальшим перерозподілом даних;
- обробка запитів на збереження даних;
- обробка запитів на читання даних;
- обробка запитів на модифікацію даних;
- обробка запитів на видалення даних;
- прозора для користувача реплікація даних;
- прозорий для користувача шардинг даних;
- вирішення конфліктів у даних на клієнтській частині;
- конфігурування рівня реплікації даних;
- задання кількості реплік, які необхідно опитати для успішного виконання запиту;
- робота в умовах розділеної мережі;
- візуальне представлення статистичної інформації про систему для моніторингу її роботи.

1.4.1 Розроблення функціональних вимог

Система передбачає наступні варіанти використання:

Таблиця 1.5 – Варіант використання UC001

Назва	Запуск на одному вузлі
Опис	Адміністратор запускає систему на одному вузлі.
Учасники	Адміністратор.
Передумови	Необхідні програмні продукти інстальовані та конфігураційний файл коректно заповнений.
Постумови	Система запущена та готова до обробки запитів.
Основний сценарій	Адміністратор запускає програму. Запускається веб сервер, що готовий приймати запити на обробку даних. Відображається повідомлення про успішний старт.
Розширення сценаріїв	

Таблиця 1.6 – Варіант використання UC002

Назва	Запуск на кількох вузлах
Опис	Адміністратор запускає систему на кількох вузлах.
Учасники	Адміністратор.
Передумови	Необхідні програмні продукти інстальовані, конфігураційний файл коректно заповнений, між вузлами є мережеве з’єднання.
Постумови	Система запущена та готова до обробки запитів.
Основний сценарій	1. Адміністратор запускає програму на кожному з вузлів. Відбувається з’єднання зі всіма вже запущеними вузлами. 2. Запускається веб сервер, готовий приймати запити на обробку даних. 3. Відображається повідомлення про успішний старт.
Розширення	

сценаріїв	

Таблиця 1.7 – Варіант використання UC003

Назва	Приєднання нового вузла
Опис	Адміністратор додає новий вузол до вже запущеного сховища.
Учасники	Адміністратор.
Передумови	Сховище запущено. На новому вузлі необхідні програмні продукти інстальовані, конфігураційний файл коректно заповнений, між вузлами є мережеве з'єднання.
Постумови	До системи приєднано новий вузол та дані перерозподілено.
Основний сценарій	1. Адміністратор запускає програму на новому вузлі. 2. Відбувається з'єднання з вузлами, на яких система вже запущена. 3. Запускається веб сервер, що готовий приймати запити на обробку даних. 4. Відбувається перерозподіл даних між вузлами. 5. Відображається повідомлення про успішний старт.
Розширення сценаріїв	

Таблиця 1.8 – Варіант використання UC004

Назва	Вилучення вузла з системи
Опис	Адміністратор вилучає вузол з вже запущеного сховища.
Учасники	Адміністратор.
Передумови	Сховище запущено.

Постумови	З системи вилучено вузол та дані перерозподілено.
Основний сценарій	1. Адміністратор зупиняє відповідний вузол. 2. Система ідентифікує відсутність вузла та перерозподіляє дані.
Розширення сценаріїв	

Таблиця 1.9 – Варіант використання UC005

Назва	Збереження даних
Опис	Клієнтський застосунок надсилає запит на збереження даних.
Учасники	Клієнтський застосунок.
Передумови	Сховище запущено.
Постумови	Дані збережено та репліковано.
Основний сценарій	1. Клієнтський застосунок надсилає запит на збереження даних за вказаним ключем. 2. До застосунку надходить відповідь з результатом виконання запиту.
Розширення сценаріїв	2.1. В наслідок несправності на запит відповіло менше реплік ніж зазначено в запиті. 2.1.а. У відповідь на запит повертається помилка, а самі дані не зберігаються.

Таблиця 1.10 – Варіант використання UC006

Назва	Читання даних
Опис	Клієнтський застосунок надсилає запит на читання даних за вказаним ключем.
Учасники	Клієнтський застосунок.
Передумови	Наявні дані за вказаним ключем.

Постумови	Дані отримано.
Основний сценарій	1. Клієнтський застосунок надсилає запит на читання даних за вказаним ключем. 2. До застосунку надходить відповідь з результатом виконання запиту.
Розширення сценаріїв	2.1. В наслідок несправності на запит відповіло менше реплік ніж зазначено в запиті. 2.1.а. У відповідь на запит повертається помилка. 2.2 За вказаним ключем немає даних. 2.2.а. У відповідь на запит повертається помилка. 2.3 При опитуванні реплік виявлені конфліктні версії даних. 2.3.а. У відповідь на запит повертається всі виявлені конфліктні версії даних.

Таблиця 1.11 – Варіант використання UC007

Назва	Модифікація даних
Опис	Клієнтський застосунок надсилає запит модифікацію даних.
Учасники	Клієнтський застосунок.
Передумови	Наявні дані за вказаним ключем.
Постумови	Дані модифіковано та репліковано.
Основний сценарій	1. Клієнтський застосунок виконує запит на зчитування даних за вказаним ключем. 2. Сховище надсилає у відповідь необхідні дані. 3. Клієнтський застосунок надсилає запит на модифікацію даних за вказаним ключем. 4. До застосунку надходить відповідь з результатом

	виконання запиту.
Розширення сценаріїв	2.1. В наслідок несправності на запит відповіло менше реплік ніж зазначено в запиті. 2.1.а. У відповідь на запит повертається помилка. 2.2. За вказаним ключем немає даних. 2.2.а. У відповідь на запит повертається помилка. 2.3. При опитуванні реплік виявлені конфліктні версії даних. 2.3.а. У відповідь на запит повертається всі виявлені конфліктні версії даних. 2.3.б. Клієнтський застосунок вирішує конфлікт та продовжує працювати за вказаним сценарієм. 3.1. Дані на основі яких відбулась модифікація застаріли. 3.1.а. У відповідь на запит повертається помилка. 3.2. В наслідок несправності на запит відповіло менше реплік ніж зазначено в запиті. 3.2.а. У відповідь на запит повертається помилка.

Таблиця 1.12 – Варіант використання UC008

Назва	Видалення даних
Опис	Клієнтський застосунок надсилає запит на видалення даних.
Учасники	Клієнтський застосунок.
Передумови	Наявні дані за вказаним ключем.
Постумови	Дані видалено.
Основний сценарій	1. Клієнтський застосунок надсилає запит на видалення даних за вказаним ключем. 2. До застосунку надходить відповідь з результатом

	виконання запиту.
Розширення сценаріїв	2.1. В наслідок несправності на запит відповіло менше реплік ніж зазначено в запиті. 2.1.а. У відповідь на запит повертається помилка, а самі дані не видаляються.

Таблиця 1.13 – Варіант використання UC009

Назва	Отримання інформації про систему
Опис	Моніторинг спеціаліст спостерігає за станом системи.
Учасники	Моніторинг спеціаліст.
Передумови	Сховище запущено.
Постумови	Стан системи відображений на веб сторінці
Основний сценарій	1. Моніторинг спеціаліст відкриває веб сторінку з інформацією про роботу системи. 2. На сторінці відображається відповідна статистична інформація.
Розширення сценаріїв	2.1. Виявлена аномалія в роботі системи. 2.1.а. Моніторинг спеціаліст сповіщає про неполадку адміністратора та програміста.

Таблиця 1.14 – Варіант використання UC010

Назва	Визначення проблемного місця в системі
Опис	Адміністратор системи та програміст визначають проблемне місце в системі.
Учасники	Адміністратор, програміст.
Передумови	Сховище запущено.
Постумови	Проблему ідентифіковано.
Основний сценарій	1. Адміністратор та програміст відкривають веб сторінку з інформацією про роботу системи.

	2. На сторінці відображається відповідна статистична інформація. 3. З допомогою отриманої інформації проблему ідентифікують.
Розширення сценаріїв	3.1. Виявлені невідомі помилки на вузлах. 3.1.а. Адміністратор та програміст переглядають логи на відповідних вузлах системи. 3.1.б. Помилки ідентифіковані.

Таблиця 1.15 – Варіант використання UC011

Назва	Прийняття рішення про масштабування системи
Опис	Адміністратор системи приймає рішення про її масштабування.
Учасники	Адміністратор.
Передумови	Сховище запущено.
Постумови	Система масштабована.
Основний сценарій	1. Адміністратор відкриває веб сторінку з інформацією про роботу системи. 2. На сторінці відображається статистична інформація по роботі системи. 3. На основі інформації про навантаження на вузли системи та кількість даних, що вони зберігають приймається рішення про масштабування системи.
Розширення сценаріїв	3.1. Помічено, що на деяких вузлах зберігається занадто велика кількість даних чи висока кількість запитів. 3.1.а. Адміністратор приймає рішення про додавання нових вузлів.

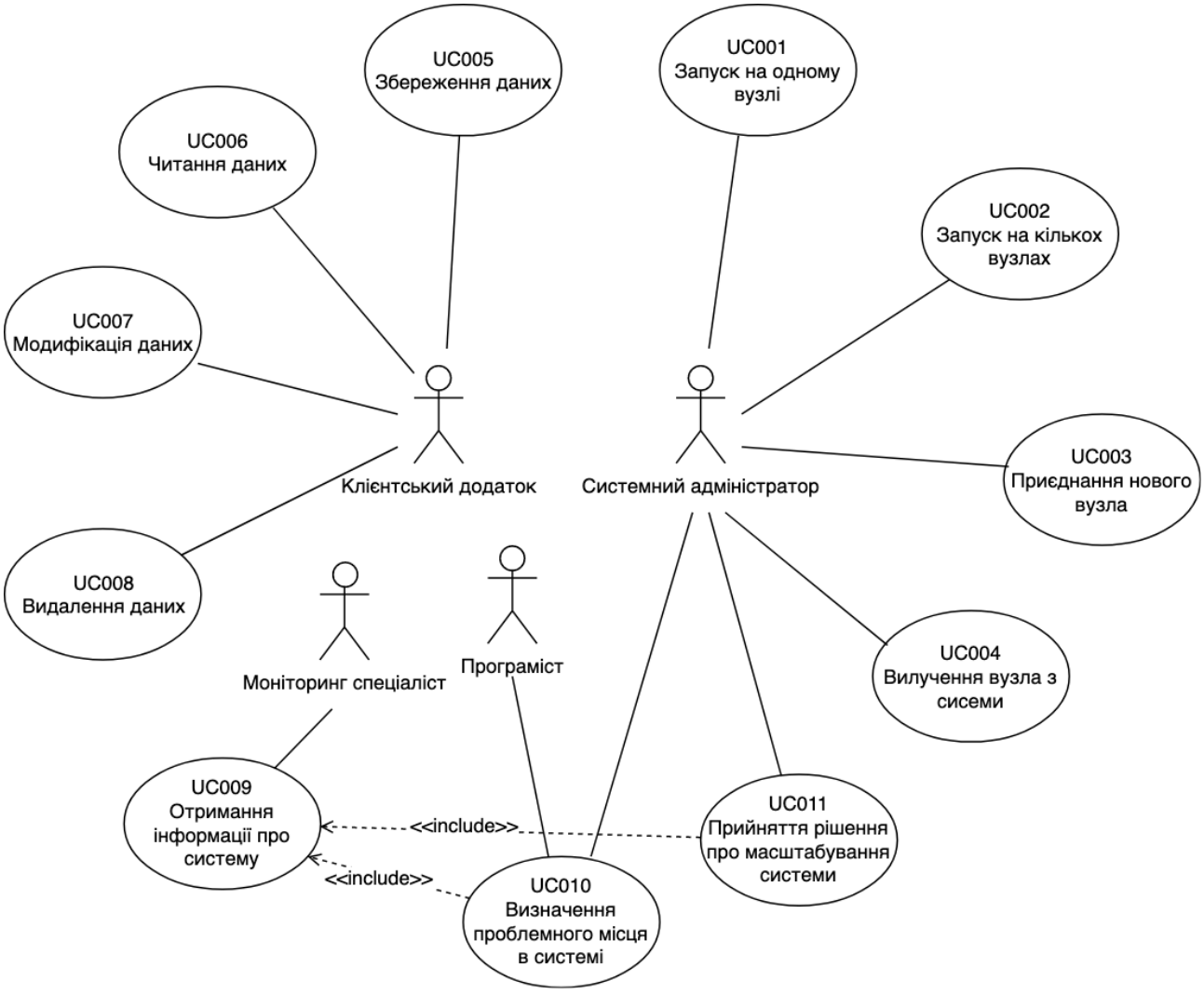

Рисунок 1.4 – Схема структурна варіантів використання

Схема варіантів використання зображена на рисунку 1.4.

Виходячи з описаних вище варіантів використання функціональні вимоги можна поділити на дві групи: функціональні вимоги до самого сховища даних та функціональні вимоги до системи моніторингу.

Функціональні вимоги до сховища даних

Вимоги до сховища даних описано наступними таблицями:

Таблиця 1.16 – Опис функціональної вимоги REQ001

Номер	REQ001
Назва	Запуск на одному та кількох вузлах
Опис	Адміністратор може запустити сховище як на одному, так і на

	кількох вузлах кластера.

Таблиця 1.17 – Опис функціональної вимоги REQ002

Номер	REQ002
Назва	Приєднання нових вузлів
Опис	Адміністратор повинен мати змогу приєднання нових вузлів до вже запущеної системи, після якої відбудеться перерозподіл даних по вузлах.

Таблиця 1.18 – Опис функціональної вимоги REQ003

Номер	REQ003
Назва	Вилучення вузлів
Опис	Адміністратор повинен мати змогу вилучення вузлів з вже запущеної системи, після якої відбудеться перерозподіл даних по вузлах.

Таблиця 1.19 – Опис функціональної вимоги REQ004

Номер	REQ004
Назва	Обробка запитів на збереження даних
Опис	Система повинна мати змогу обробляти запити на збереження даних за вказаним ключем.

Таблиця 1.20 – Опис функціональної вимоги REQ005

Номер	REQ005
Назва	Обробка запитів зчитування даних
Опис	Система повинна мати змогу обробляти запити на зчитування

	даних за вказаним ключем.

Таблиця 1.21 – Опис функціональної вимоги REQ006

Номер	REQ006
Назва	Обробка запитів видалення даних
Опис	Система повинна мати змогу обробляти запити на видалення даних за вказаним ключем.

Таблиця 1.22 – Опис функціональної вимоги REQ007

Номер	REQ007
Назва	Обробка запитів модифікації даних
Опис	Система повинна мати змогу обробляти запити на модифікацію даних за вказаним ключем.

Таблиця 1.23 – Опис функціональної вимоги REQ008

Номер	REQ008
Назва	Реплікація даних
Опис	Система повинна прозоро для користувача проводити реплікацію даних за вказаним в конфігураціях рівнем реплікації.

Таблиця 1.24 – Опис функціональної вимоги REQ009

Номер	REQ009
Назва	Шардінг даних
Опис	Система повинна прозоро для користувача проводити шардінг даних рівномірно розподіляючи дані по всіх вузлах.

Таблиця 1.25 – Опис функціональної вимоги REQ010

Номер	REQ010
Назва	Задання рівня реплікації даних
Опис	Рівень реплікації даних повинен налаштовуватись при первинній конфігурації системи.

Таблиця 1.26 – Опис функціональної вимоги REQ011

Номер	REQ011
Назва	Задання мінімальної кількості реплік для виконання запиту
Опис	У кожному запиті до сховища програміст повинен мати можливість вказати кількість реплік, що повинні відповісти для успішного виконання запиту.

Таблиця 1.27 – Опис функціональної вимоги REQ012

Номер	REQ012
Назва	Вирішення конфліктів у даних
Опис	При виникненні конфліктів у даних при одночасній їх модифікації система повинна надавати користувачу змогу вирішити даний конфлікт, щоб уникнути втрат в даних.

Функціональні вимоги до системи моніторингу

Вимоги до системи моніторингу описано наступними таблицями:

Таблиця 1.28 – Опис функціональної вимоги REQ013

Номер	REQ013
Назва	Відображення кількості запитів
Опис	Система моніторингу повинна відображати кількість запитів на зчитування, запис, модифікацію та видалення даних на

	кожному вузлі окремо.

Таблиця 1.29 – Опис функціональної вимоги REQ014

Номер	REQ014
Назва	Відображення об’єму даних
Опис	Система моніторингу повинна відображати об’єм даних, що зберігається на кожному вузлі окремо, це дозволить визначити простоюючі та перевантажені вузли.

Таблиця 1.30 – Опис функціональної вимоги REQ015

Номер	REQ015
Назва	Відображення кількості помилок
Опис	Система моніторингу повинна відображати кількість помилок на кожному з вузлів окремо, що дозволить ідентифікувати проблемні вузли.

Структура функціональних вимог зображена на рисунку 1.5.

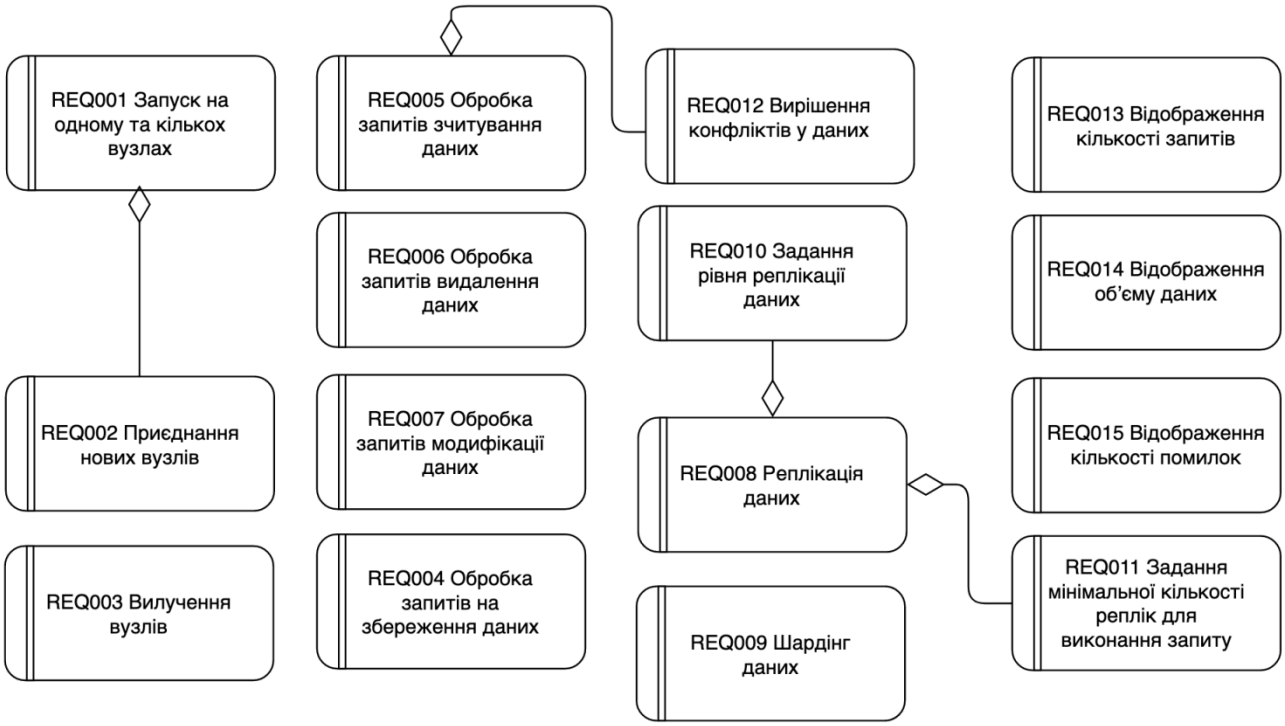

Рисунок 1.5 – Схема структурна залежності вимог

Матриця трасування між функціональними вимогами та варіантами використання зображена на рисунку 1.6.

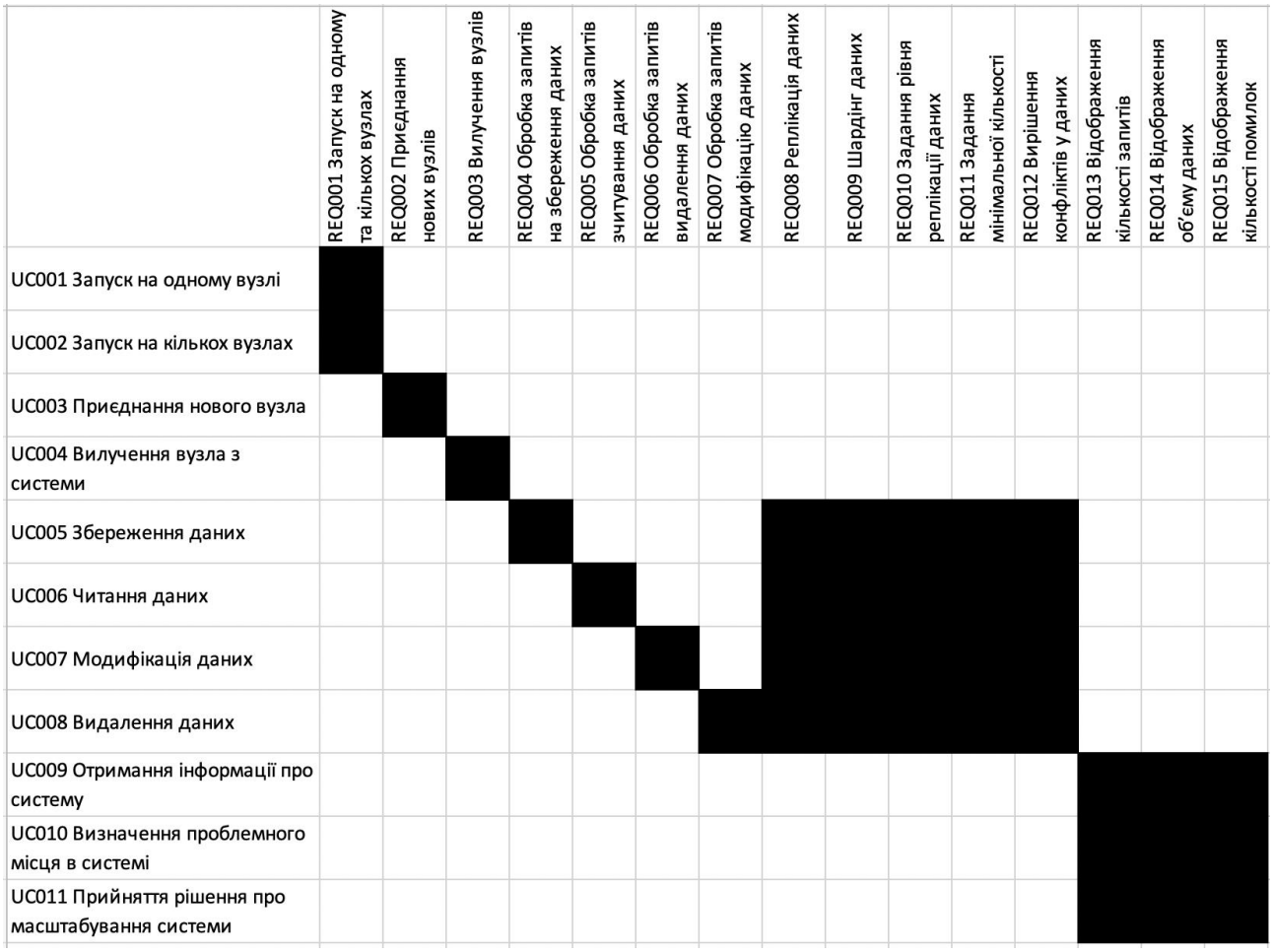

Рисунок 1.6 – Схема структурна матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Програмне забезпечення має відповідати наступним нефункціональним вимогам:

– взаємодія зі сховищем повинна бути організована у вигляді json запитів;

– для передачі даних повинні використовуватись захищені канали передачі даних з використанням TLS;

– доступ до даних повинен надаватись через будь-який з вузлів системи;

– за умов розділу мережі на дві та більше окремі частини кожен вузол системи повинен продовжувати обробляти запити, а при відновленні зв'язку між ізольованими частинами системи, конфлікти у даних повинні бути розв'язані, а самі дані коректно репліковані;

– час відклику системи повинен бути мінімальним, у межах кількох мілісекунд;

– дані в системі повинні рівномірно розподілятись по вузлам.

1.4.3 Постановка комплексу завдань модулю

Програмне забезпечення, що розробляється, призначене для надійного зберігання та обробки великих об’ємів даних, до яких необхідний оперативний доступ.

Мета створення даної роботи – створення розподіленого сховища даних, що на відміну від більшості сучасних рішень, дозволяє динамічно змінювати кількість вузлів та попри модель узгодженості даних з часом, дозволяє вирішувати конфлікти при конкурентній модифікації даних на стороні клієнта, що дозволяє уникнути втрат даних.

Для досягнення даної мети система повинна вирішувати наступні задачі:

– приєднання нових вузлів з подальшим перерозподілом даних по них;

– вилучення вузлів з системи з подальшим перерозподілом даних;

– обробка запитів на збереження даних;

– обробка запитів на модифікацію даних;

– обробка запитів на видалення даних;

– прозора для користувача реплікація даних;

– прозорий для користувача шардинг даних;

– вирішення конфліктів у даних на клієнтській частині;

– конфігурування рівня реплікації даних;

– задання кількості реплік, які необхідно опитати для успішного виконання запиту;

– робота в умовах розділеної мережі;

– візуальне представлення статистичної інформації про систему для моніторингу її роботи.

Для роботи з сховищем передбачена система моніторингу, що надає статистичну інформацію про роботу системи.

1.5 Висновки по розділу

Збереження даних у розподілених сховищах має як позитивні так і негативні сторони. З одного боку це дозволяє обробляти практично необмежені об’єми даних, динамічно горизонтально масштабувати систему для при зміні навантаження на неї, забезпечувати надійність, швидкодію та відмовостійкість. Проте це зазвичай потребує відмови від ACID принципів, приводить до появи неузгоджених даних та змушує змінювати представлення доменної області, через обмеження в таких сховищах.

Всі наявні рішення пропонують певний компроміс між цими перевагами та недоліками, однак він не завжди підходить для вирішення деяких задач. Так жодна з проаналізованих систем не надає гарантії у збереженні всіх модифікацій даних в умовах розділу мережі, так як системи, що можуть працювати в таких умовах використовують найпростіший механізм вирішення конфліктів за останньою часовою міткою, що супроводжується можливими втратами в даних. Обробка конфліктів на клієнтській стороні дозволяє обійти це обмеження, забезпечуючи не тільки уникнення втрат даних, а й забезпечити високу швидкодію такої системи, хоч і потребує більше зусиль від користувача даної системи.

Розробка системи, що містить такий унікальний набір характеристик, дозволить у подальшому використовувати її при розробці високонавантажених систем, у яких окрім постійної доступності, також є вимога до відсутності втрат в даних.

Розподілене сховище повинне рівно розподіляти дані між вузлами та проводити їх реплікацію. При зміні кількості вузлів, воно має автоматично перерозподіляти дані по системі. Кожен вузол систему повинен мати змогу обробляти запити на модифікацію та зчитування даних у будь-який момент.

Для вводу такої системи в експлуатацію необхідно, щоб вона містила підсистему моніторингу, що б дозволяла слідкувати за її роботою, та дозволяла б приймати рішення про необхідність масштабування.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для створення програмного продукту необхідно провести моделювання процесів, що протікають у ньому та його архітектури у цілому. Для цього було обрано метод побудови BPMN діаграм. Даний спосіб представлення був вибраний, так як система, що розглядається є кластерною, а отже потребує зображення не лише алгоритмічної частини, а й послідовності взаємодій між вузлами системи.

Для розгляду архітектури розроблюваного розподіленого сховища даних розглянемо основні типи сценаріїв, що в ньому протікають:

- обробка запиту на збереження даних;
- обробка запиту на зчитування даних;
- обробка запиту на модифікацію даних;
- обробка запиту на видалення даних;
- приєднання нового вузла;
- вилучення вузла;
- обробки конфліктних ситуацій.

Кожен з цих процесів може працювати одночасно з іншими, змінюючи спільний стан системи, що суттєво ускладнює її розробку, відладку та тестування.

Для спрощення діаграм, на них не відображаються дії пов’язані з підсистемою моніторингу, однак вони є тривіальними та не потребують окремого пояснення.

Клієнт сховища може надіслати запит на будь-який вузол сховища. Така поведінка дозволяє ефективно балансувати навантаження між усіма вузлами з допомогою незалежного балансувальника трафіку, що не повинен мати представлення про внутрішню структуру сховища.

Вузол, на який потрапив запит, що на схемі позначено як приймаючий вузол. Так як всі вузли системи знаходяться в межах одного кластеру, вони мають доступ до стану кластеру, який прозоро для користувача поширюється на всі вузли. Стан кластеру розповсюджується з допомогою gossip протоколу, та являє собою набір вільні від конфліктів структур даних, що містить інформацію про стан вузлів та відповідність між вузлами системи та даними, які на них зберігаються.

Так як для горизонтальної фрагментації даних в системі використовується алгоритм узгодженого хешування, мітки вузлів повинні бути збережені в стані кластеру, для доступу до них з будь-якого вузла. Таким чином для визначення вузла, що відповідає за збереження даних необхідно лише знайти значення хеш функції від ключа даних, після чого знайти перший вузол мітка, якого знаходиться дані за значенням отриманого хешу. Цей вузол назвемо координуючим.

Окрім фрагментації в сховищі також застосовується механізм реплікації. Рівень реплікації даних вказується при початковій настройці сховища, однак при виконанні кожного з запитім є можливість вказати кількість реплік, відповіді від яких слід очікувати для успішного завершення запиту, що дозволяє обирати між синхронною та асинхронною реплікацією. В якості реплік використовуються наступні R - 1 вузлів, мітки яких слідують за міткою координуючого вузла, де R – рівень реплікації.

Для початку розглянемо загальну архітектуру сховища на прикладі процесу обробки запиту на зчитування даних за вказаним ключем. Схему його роботи зображено на рисунку 2.1.

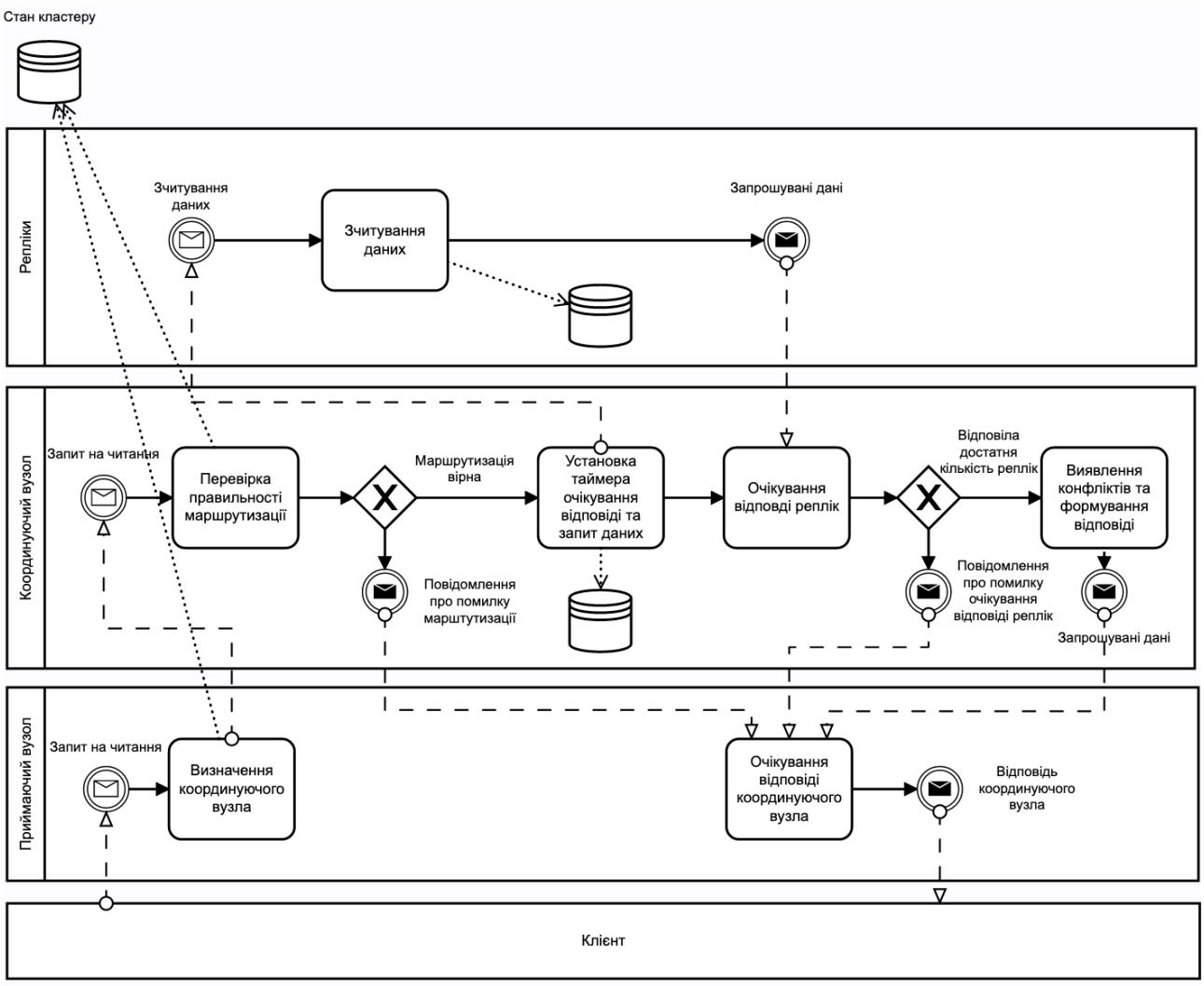

Рисунок 2.1 – Схема структурна виконання запиту на читання даних

При зчитуванні даних клієнт робить запит на будь-який вузол, після чого запит вже перенаправляється на координуючий вузол. Для цього приймаючий вузол отримує стан кластеру, визначає за координуючий вузол за алгоритмом узгодженого хешування, та надсилає йому запит на читання.

На координуючому вузлі знову відбувається запит стану кластеру, а також для визначення списку реплік, для того, щоб перевірити правильність маршрутизації, так як за час пересилки запиту конфігурація кластеру могла змінитись. При виявленні помилки, вона відправляється на приймаючий вузол та пересилається клієнту.

Після перевірки правильності маршрутизації координуючий вузол установлює таймер для очікування відповіді від реплік, та запрошує дані у

реплік, при цьому також роблячи запит у власну базу даних. На репліках запит на зчитування обробляється, та надсилаються на координуючий вузол.

Координуючий вузол очікує на відповіді реплік зазначений час, якщо відповіла мінімальна кількість реплік, що вказана в запиті, запит вважається успішним, в іншому випадку приймаючому вузлу, а отже і клієнту надсилається помилка очікування на відповідь реплік.

У випадку, коли мінімальна кількість реплік, що вказана в запиті рівна одиниці, координуючий вузол робить запит тільки до локальної бази, та одразу надсилає відповідь клієнту.

Після отримання відповідей від реплік, з допомогою порівняння векторних годинників визначаються конфліктні версії даних, якщо їх не виявлено, то повертається тільки найсвіжіша версія, в іншому випадку повертаються всі найсвіжіші конфліктуючі версії.

Результат обробки конфліктів пересилається приймаючому вузлу, який в свою чергу відправляє його клієнту.

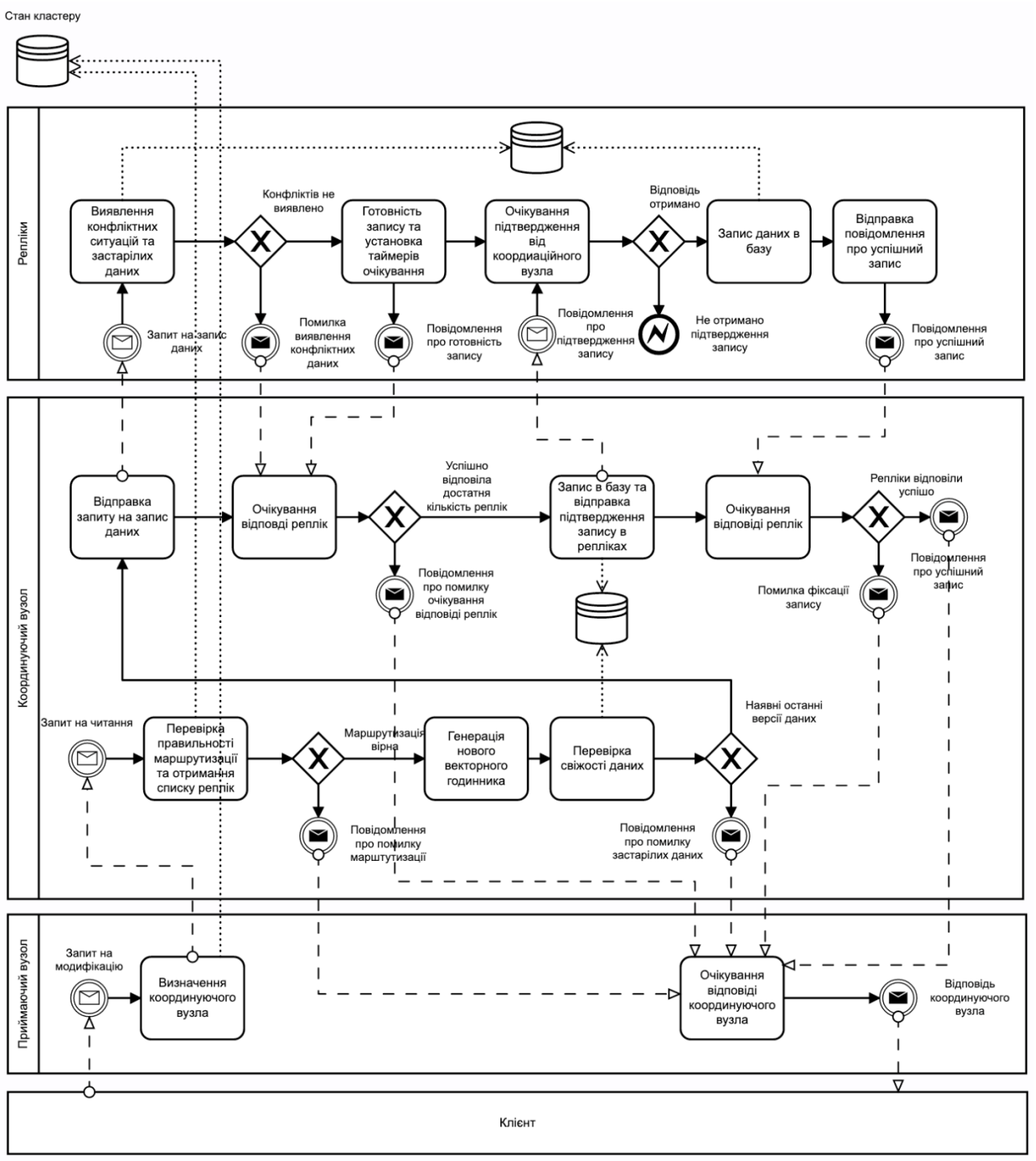

Рисунок 2.2 – Схема структурна виконання запиту на модифікацію даних

Тепер розглянемо модифікацію даних в сховищі. Це набагато складніший процес, так як необхідно надійно реплікувати їх.

Для початку, як і у випадку з читанням, клієнт робить запит на будь-який вузол системи. Після цього приймаючий вузол запрошує стан кластеру та

визначає який з вузлів є координуючим для даних, що зберігаються за даним ключем, та пересилає запит на нього.

Отримавши запит, координуючий вузол на основу стану кластера визначає чи маршрутизація пройшла успішно та знаходить список реплік. Якщо виявлена помилка, вона пересилається приймаючому вузлу, що пересилає її клієнту.

Якщо маршрутизація вірна, обробка запиту продовжується і для даних генерується новий векторний годинник шляхом інкрементації компонента відповідаючого за координуючий вузол векторного годинника даних, що ж прообразом модифікованих даних. Отримавши нову версію, її порівнюють з наявною версією даних на координуючому вузлі. У випадку виявлення конфлікту надсилається повідомлення про помилку.

Після цього відбувається запис на репліки з використанням механізму двох крокової фіксації. Для цього спершу дані відправляються на репліки, там вони перевіряються на відсутність конфліктів, та у випадку відсутності проблем повідомляють координаційний вузол.

Якщо координаційний вузол отримав позитивні відповіді від необхідної кількості реплік, він посилає повідомлення про підтвердження запису модифікації реплікам, в свою чергу записуючи дані в локальну базу даних.

Отримавши це повідомлення, репліки виконують запис в свої бази, та відсилають повідомлення про успішний запис координаційному вузлу.

Координаційний вузол отримавши необхідну кількість позитивних відповідей надсилає приймаючому вузлу повідомлення про завершення модифікації, який пересилає його клієнту. При виникненні будь-якої помилки клієнт також сповіщається.

При початковому збереженні даних, процес такий же, як і при їх модифікації даних, однак в якості версії прообразу даних використовується пустий векторний годинник, тобто векторний годинник, у якому зі всіма вузлами асоційована версія 0.

Для видалення даних також використовується механізм модифікації, однак в якості значення записується пусте посилання, що означає відсутність даних.

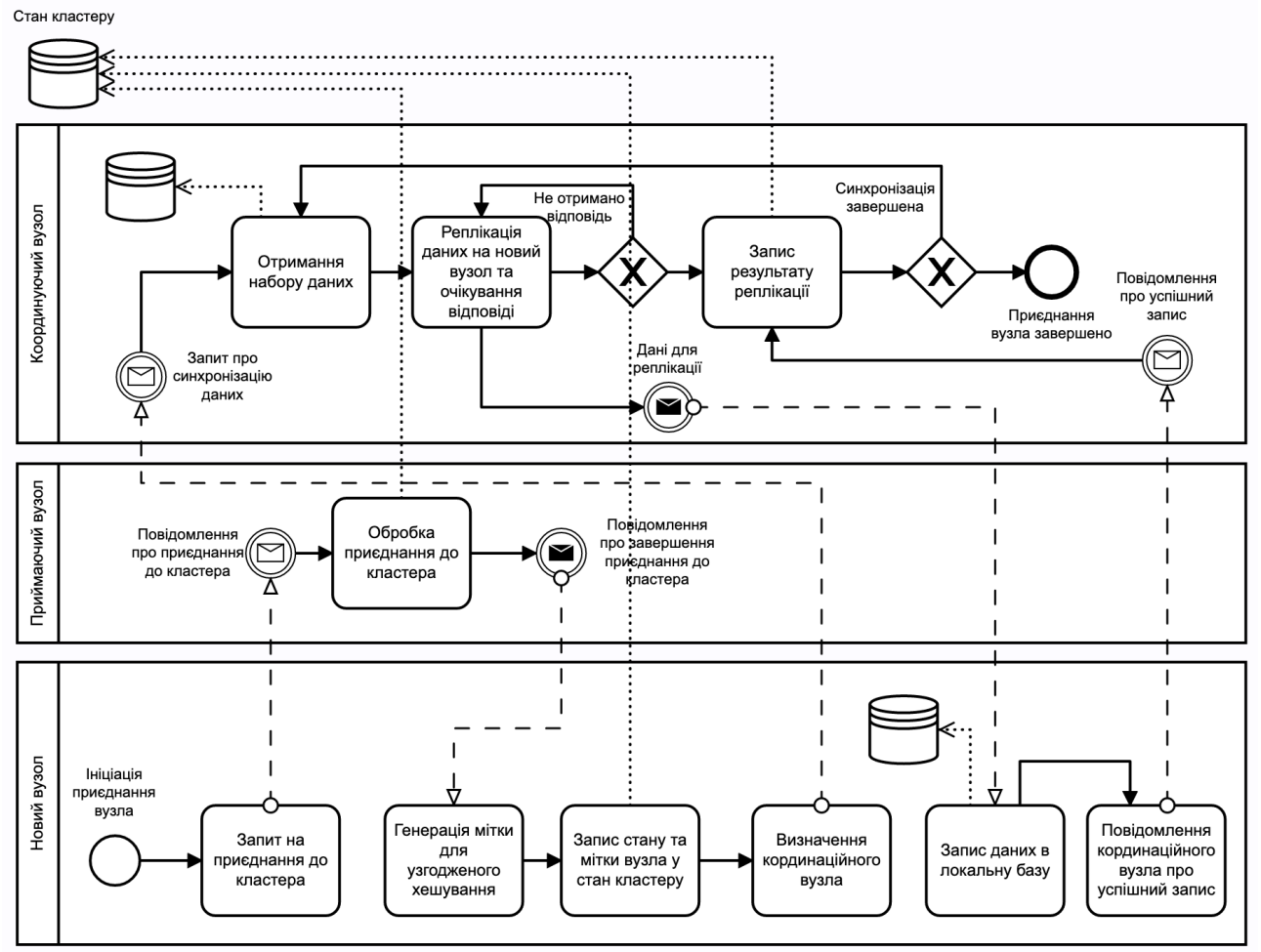

Рисунок 2.3 – Схема структурна долучення вузла до кластера

Тепер слід розглянути додавання вузла до системи. Діаграма цього процесу зображена на рисунку 2.3. Спершу новий вузол виконує запит до одного з вузлів кластеру, приймаючий вузол обробляє його, повідомляє всі вузли про долучення та надсилає новому вузлу повідомлення про завершення долучення.

Після цього новий вузол генерує мітку для узгодженого хешування та записує її у стан кластеру. Тоді визначається координаційний вузол даних, що необхідно перемістити на новий вузол.

Новий вузол посилає запит на синхронізацію даних координаційному вузлу, який вибирає частину даних, що потребують синхронізації, та посилає їх на новий вузол, де вони записуються в локальну базу даних. Новий вузол сигналізує координаційний про завершення запису, а координаційний вузол записує прогрес синхронізації в стан кластеру, після чого процес повторюється до тих пір, поки всі дані не будуть синхронізовані.

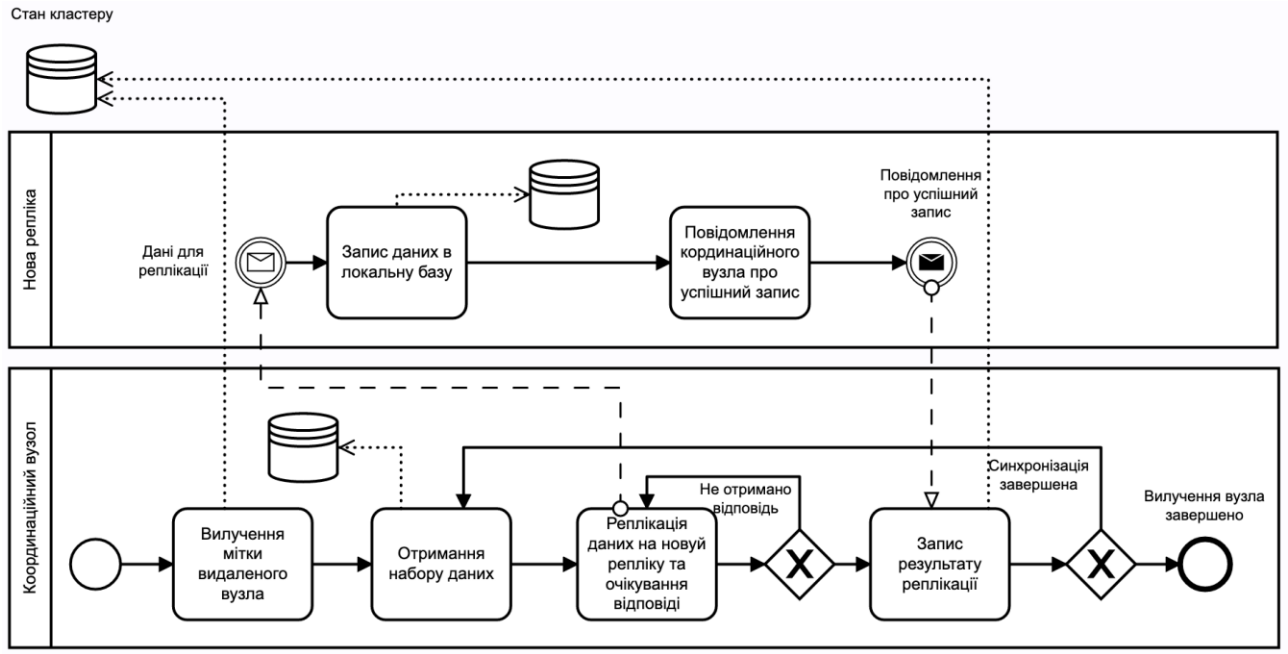

Рисунок 2.4 – Схема структурна вилучення вузла з кластера

При вилученні вузла з кластера процес достатньо схожий, однак процес дещо простіший, так як перерозподіл даних ініціює координуючий вузол ініціює процес. Він видаляє мітку видаленого вузла з стану кластера, та визначає новий вузол, що виконуватиме його роль. Після чого як і при додаванні вузла, відбувається процес синхронізації даних. Цей процес показано на рисунку 2.4.

2.2 Архітектура програмного забезпечення

Розроблюване програмне забезпечення реалізоване як розподілений серверний додаток, що накладає певні обмеження на його архітектуру.

Перш за все необхідно, щоб додаток мав змогу працювати у кластерному режимі. Це не проста вимога, так як це потребує постійної комунікації між усіма вузлами кластеру, виявлення нови та вилучення несправних вузлів, підтримки стану кластера та розповсюдження його по всіх вузлах.

Іншою проблемою є вимога до високої швидкодії, так як в логіці додатку передбачена велика кількість мережевих взаємодій. При звичайному підході з синхронними взаємодіями між компонентам знадобиться велика кількість потоків виконання, що просто простоюватимуть в очікуванні на відповідь вузлів системи. Це може суттєво знизити швидкодію сховища навіть при невеликому навантаженні. Саме тому для цієї задачі необхідно застосовувати підходи асинхронного програмування.

Через ці два обмеження було обрано мову програмування Scala. Ця мова програмування підтримує різні парадигми та має чудову підтримку асинхронного програмування. [9] Крім цього Scala є сильно типізованою мовою, що суттєво спрощує написання великих програм з її використанням, так як чимало помилок відловлюються під час компіляції. Scala виконується з допомогою віртуальної машини Java, таким чином програми, написані на ній не тільки можуть використовувати всі Java бібліотеки, а й можуть виконуватись на будь-якій платформі, що підтримує Java.

Для забезпечення взаємодії між компонентами системи в кластері та асинхронного програмування було обрано бібліотеку Akka. В її основі лежить модель акторів. Це математична модель паралельних обчислень, основним поняттям якої є актор – універсальний примітив паралельного виконання. Актори взаємодіють між собою з допомогою відправки повідомлень. Отримані повідомлення додаються в чергу, так звану поштову скриньку, та дістаються актором по мірі його звільнення. При обробці повідомлення, актор може надіслати повідомлення іншому актору, створити нового актора та змінювати поведінку обробки наступних повідомлень. [10] Детальний розгляд акторної моделі виходить за межі даної роботи.

Akka має модульну структуру та містить наступні основні модулі:

– Actors – відповідає за роботу з акторами;

– Remote – дозволяє працювати з акторами, що знаходяться на інших вузлах системи прозоро, не знаючи їх реального розміщення;

– Http – містить http-сервер та клієнт для мережевої взаємодії з іншими компонентами;

– Clustering – робота у кластерному режимі;

– Streams – обробка потокових даних;

Як видно, Akka містить потужний набір інструментів, що ідеально підходить для написання розподіленого сховища даних.

Передбачено, що кожен вузол системи містить власну локальну базу даних. Так як доступ до цієї бази відбуватиметься лише з того ж вузла, локальній базі не потрібний інтерфейс для мережевої роботи, тож оптимальним рішенням є застосування вбудованої бази даних. Попри це дуже важливим критерієм є її швидкодія, та можливість паралельного виконання запитів. Саме тому було обрано реляційну базу H2. Вона відповідає всім вище переліченим вимогам.

В якості системи моніторингу було обрано найпопулярнішу на даний момент систему – Prometheus. Дана система передбачає, що додаток буде надавати значення відповідних метрик у стандартизованому форматі при HTTP запиті за вказаним адресом. Інтервал, з яким Prometheus буде отримувати значення метрик задається при налаштуванні систем. Отримані дані від зберігає у спеціально розробленій для цього базі даних. Prometheus підтримує власну мову запитів, що дозволяє отримувати значення метрик за вказаний часовий період, проводити над ними перетворення та представляти їх у різних формах.

Щоб представляти дані, зібрані Prometheus використано програмний продукт Grafana, який дозволяє зручно візуалізувати дані у вигляді графіків, лічильників, гістаграм та інших графічних представлень. Даний продукт також являється серверним додатком, та надає користувачу інформацію на веб

сторінці. Приклад візуалізації даних з допомогою Grafana показано на рисунку 2.5.

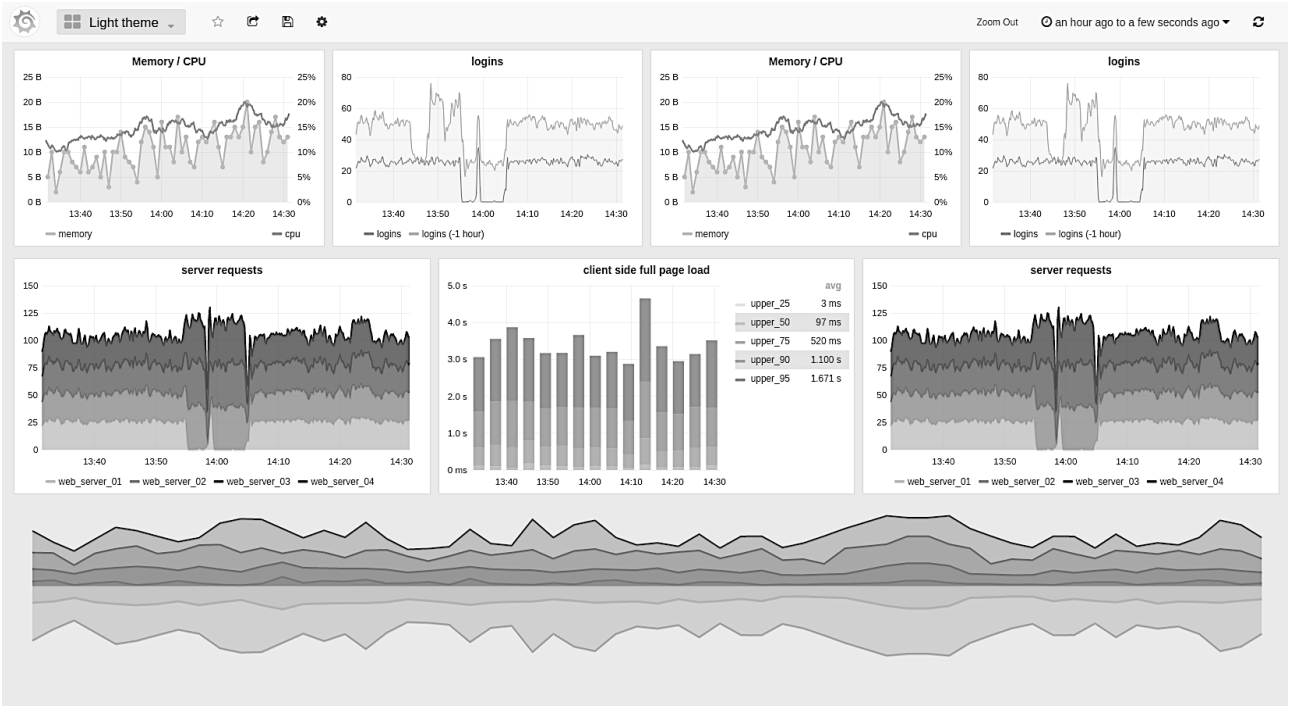

Рисунок 2.5 – Візуалізація метрик з використанням Grafana

2.3 Конструювання програмного забезпечення

Так як для розробки використовується модель акторів, слід виділити типи акторів, що є в системі, типи повідомлень, якими вони обмінюються.

Основні актори, їх опис та типи повідомлень, що вони можуть обробляти вказані в таблиці 2.1.

Таблиця 2.1 – Актори системи

Актор	Опис	Повідомлення, що обробляються
StorageController	Відповідає за обробку HTTP запитів.	– GetReq; – PutReq; – ModReq; – DelReq.
DataReplicator	Копіює дані на репліки,	– Replicate;

	обробляє їх відповіді.	– ReplicationFailure; – ReplicationReady; – ReplicationDone.
StorageManager	Займається безпосередньою взаємодією з локальним сховищем та бізнес логікою по обробці даних.	– Get; – Put; – Mod; – Del.
RequestRouter	Відповідає за маршрутизацією запитів до координаційоного вузла.	– Get; – Put; – Mod; – Del.
ShardManager	Відповідає за розподіл даних по відповідних шардах, обробляє перерозподіл даних.	– MemberAdded; – MemberRemoved.

Наявні в системі повідомлення, та дані, що вони містять вказано в таблиці 2.2.

Таблиця 2.2 – Повідомлення в системі

Повідомлення	Опис	Данні
GetReq	Повідомлення, що повторює структуру JSON запиту на зчитування даних	– кількість реплік для зчитування; – ключ.
PutReq	Повідомлення, що повторює структуру JSON запиту на запис даних	– кількість реплік для запису; – ключ; – значення.
ModReq	Повідомлення, що повторює структуру JSON запиту на	– кількість реплік для запису;

	модифікацію даних	– версія прообразу даних; – ключ; – нове значення.
DelReq	Повідомлення, що повторює структуру JSON запиту на видалення даних	– кількість реплік для запису; – версія прообразу даних; – ключ.
Replicate	Повідомлення з командою реплікаціх даних	– кількість реплік для запису; – ключ; – значення.
ReplicationFailure	Повідомлення про помилку реплікації	– повідомлення про помилку;
ReplicationReady	Повідомлення про готовність репліки до запису даних	
ReplicationDone	Повідомлення про успішний запис даних на репліку	
Get	Запит на зчитування даних	– кількість реплік для зчитування; – ключ.
Put	Запит на запис даних	– кількість реплік для запису; – ключ; – значення.
Mod	Запит на модифікацію даних	– кількість реплік для запис;

		– версія прообразу даних; – ключ; – нове значення.
Del	Запит на видалення даних	– кількість реплік для запису; – версія прообразу даних; – ключ.
MemberAdded	Повідомлення про долучення нового вузла в систему	– ідентифікатор вузла.
MemberRemoved	Повідомлення про вилучення вузла з системи	– ідентифікатор вузла.

Як описано вище, для збереження даних використовується реляційна база даних H2. Основна таблиця, що міститиме усі збережені дані, матиме структуру, описану в таблиці 2.3.

Таблиця 2.3 – Опис таблиці data

Опис таблиці				
Назва	data			
Опис	Данні, що зберігаються			
Опис стовбців таблиці				
Назва	Опис	Тип даних	Обов’язкове	Первинний ключ
key	Ключ даних, що зберігаються	blob	X	X
value	Данні, що зберігаються	blob		

hash	Хеш ключа для сортування даних	int	X	
clock	Векторний годинник для виявлення конфліктів	json	X	

Стовбець clock являється json об’єктом, у якого ключі – це ідентифікатори вузлів, а значення – відповідні цілочисельні версії.

Самі запити до сховища до сховища використовують в json форматі, та мають поля, описані в наступних таблицях.

Таблиця 2.4 – Формат запиту зчитування даних

Параметр	Тип даних	Опис
type	“get”	Ідентифікує запит як запит на читання даних.
key	string	Ключ даних, що зчитуються.
replicas	integer	Кількість реплік, відповідь від яких слід отримати, щоб успішно завершити запит.

Таблиця 2.5 – Формат запиту запис даних

Параметр	Тип даних	Опис
type	“put”	Ідентифікує запит як запит на запис даних.
key	string	Ключ даних, що записуються.
value	string	Значення, що зберігається.
replicas	integer	Кількість реплік, відповідь від яких слід отримати, щоб успішно завершити

		запит.

Таблиця 2.6 – Формат запиту модифікацію даних

Параметр	Тип даних	Опис
type	“mod”	Ідентифікує запит як запит на модифікацію даних.
key	string	Ключ даних, що модифікуються.
value	string	Нове значення.
clock	object	Значення векторного годинника, що асоційований з прообразом модифікованих даних. Ключами є ідентифікатори вузлів, а значенням – відповідні цілочисельні версії.
replicas	integer	Кількість реплік, відповідь від яких слід отримати, щоб успішно завершити запит.

Таблиця 2.7 – Формат запиту видалення даних

Параметр	Тип даних	Опис
type	“del”	Ідентифікує запит як запит на видалення даних.
key	string	Ключ даних, що видаляються.
clock	object	Значення векторного годинника, що асоційований з прообразом модифікованих даних. Ключами є ідентифікатори вузлів, а значенням – відповідні цілочисельні версії.
replicas	integer	Кількість реплік, відповідь від яких слід

		отримати, щоб успішно завершити запит.

У відповідь на запит можуть повернутись як данні, так і помилка, тому її структура описана в наступній таблиці.

Таблиця 2.8 – Формат відповіді на запити до сховища

Параметр	Тип даних	Опис
values	array of strings	Отримані значення. Якщо даних немає, повертається пустий масив. При відсутності конфліктів – повертається масив з одним елементом. У разі виявлення конфліктів – повертаються всі конфліктні версії. У разі виявлення помилки – не надсилається.
clock	object	Значення векторного годинника, що асоційований з отриманими даним. У разі виявлення помилки – не надсилається.
message	string	Повідомлення про помилку. У разі відмітності помилки – не надсилається.

2.4 Аналіз безпеки даних

Для забезпечення безпеки даних, зв'язок між вузлами системи, а також між вузлами та клієнтами повинен відбуватись з використанням протоколу TLS. Доступ до вузлів системи варто максимально обмежувати шляхом ізоляції їх в закритій підмережі, що унеможливлює несанкціонований доступ до них.

При використанні описаних вище методів, несанкціонований доступ до даних не є можливим.

2.5 Висновки по розділу

У цьому розділі було розглянуто технічну реалізацію системи. Для цього було проведено її моделювання з використанням нотації BPMN. Моделювання допомогло визначити всі процеси, що необхідні для повноцінного функціонування розподіленого сховища та обрати оптимальну технологію для його реалізації.

Окрім основної технології, було обрано рішення для моніторингу системи, що агрегує метрики, та дозволяє їх аналізувати, а також систему для їх візуалізації.

Після цього були описані основні актори, що взаємодіють в системі, а також повідомлення, які ці актори обробляють.

Останнім кроком став опис структури сховища, та його API.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Тестування програмного забезпечення є невід'ємною частиною циклу його розробки, так як воно гарантує правильність його реалізації, наявність всіх заявлених можливостей та відсутність помилок.

Існує велика кількість підходів до тестування. Воно може бути як виконано вручну, так і автоматизоване. Автоматизоване тестування суттєво спрощує процес розробки для програміста, так як воно гарантує, що його зміни в коді не привели до припинення роботи вже реалізованого функціоналу, дозволяючи впевнено та швидко вносити зміни в кодову базу.

При розробці розподіленого сховища даних, тестування є надзвичайно важливим, адже розподілені системи часто містять нетривіальну логіку, а також в них великий шанс виникнення помилок в наслідок конкурентної обробки інформації. До того ж, даний програмний продукт повинен коректно реагувати на виникнення мережених помилок, неполадок на самих вузлах системи, та проблеми з узгодженістю даних, крім цього працювати при великих навантаженнях та з великими об'ємами даних. В таких умовах тестування є надзвичайно складним, однак не неможливим.

Ключовими типами тестування даного програмного продукту є навантажувальне тестування, а також інтеграційне та системне тестування. Це обумовлено великою кількістю асинхронних взаємодій при його роботі, розподіленою природою продукту та умовами експлуатації.

Навантажувальне тестування в даному випадку дозволить перевірити не тільки швидкодію сховища, а й коректність алгоритмів реплікації та фрагментації даних, що можуть бути чутливими до навантаження на систему.

Таким чином будуть протестовані наступні функції системи:

– запис даних;

– читання даних;
– модифікація даних;
– видалення даних;
– реплікація даних;
– фрагментація даних;
– робота в умовах розділеної мережі;
– долучення вузлів;
– вилучення вузлів;
– відображення метрик.

Тестування цих функцій дозволить повністю перевірити отриманий програмний продукт на відповідність функціональним вимогам.

3.2 Опис процесів тестування

Тестування відбуватиметься у режимі сірої коробки. Так як, у даному продукті дуже важлива и□ого робота при непередбачуваних ситуаціях та відмовостійкість, функціонал буде протестовании□ як при позитивних, так і при негативних умовах. Так як велику частину функціоналу продукту складно перевірити маючи доступ звичаи□ного користувача, при тестуванні застосовуватимуться маніпуляції□ з серверами, на яких продукт запускається, що опосередковано, з допомогою знань про внутрішню будову продукту, дозволять протестувати дании□ функціонал.

Будуть проводитись наступні типи тестів:

– димне тестування роботи API сховища;
– критичне тестування досяжності даних при додаванні та видаленні серверів;
– системне тестування компонентів сховищ;
– інтеграціи□не тестування компонентів сховища;
– критичне тестування часу відповіді на запити;
– розширене тестування вказування рівня узгодженості при запитах;

- розширення тестування коректності показників;
- навантажувальне тестування сховища.

Навантажувальне тестування є одним з наи□важливішим для даного застосунку, тому и□ого проведенню потрібно приділити особливу увагу. Навантажувальне тестування повинно входити як до етапу регресіи□ного тестування при введені нового функціоналу, так і до приймального тестування, адже у вимогах до системи є чіткі вимоги по швидкодії□. Для проведення навантажувального тестування використовується інструмент JMeter. Він є найбільш розповсюдженим інструментом для такого тестування. JMeter симулює роботу заданої кількості клієнтів, що роблять запити до програмного продукту з вказаною частотою, а також має підтримку написання складних сценарії для тестів та дозволяє генерувати звіти щодо швидкодії системи.

Для ручного тестування базового функціоналу сховища застосовується інструмент під назвою Postman. Він дозволяє здійснювати HTTP запити з вказаними параметрами, та надає зручний інтерфейс для роботи з ними. Завдяки ньому можна перевірити коректність алгоритмів приєднання та вилучення вузлів, що достатньо складно повністю автоматизувати.

3.3 Опис контрольного прикладу

Повний опис процесу тестування та наявних сценаріїв для тестування вказано в додатку В “Програма та методика тестування”, а в даному розділі розглянуто проведення тестування на прикладі сценарію перевірки коректності зчитування даних. Його опис вказано в таблиці 3.1.

Таблиця 3.1 – Перевірка роботи зчитування даних

Мета тесту	Перевірка роботи запитів зчитування даних.
Початковий стан	В системі записані вхідні дані, запит на запис даних був обробленим вузлом 0.

	Кількість вузлів у системі 3. Рівень реплікації даних 2.
Вхідні дані	Ключ: foo Значення: bat Векторний годинник: { “0”: 1 } Кількість вузлів, відповідь від яких очікується: 2
Схема проведення тесту	Зробити запит до сховища з даними вказаними у вхідних даних, та перевірити, що у відповідь надійдуть очікувані дані.

Очікуваний результат	У відповідь на запит буде отримано наступний json об’єкт:
	{
	“values”: [“bar”],
	“clock”: { “0”: 1 }
	}

Стан програмного продукту після проведення випробування	Після проведення даного тесту стан системи не повинен змінитись.

Розглянутий тест відноситься до тестів базової функціональності та виконується з допомогою утиліти Postman. Для його виконання слід запустити сховище у конфігурації з трьох серверів та рівнем реплікації 2. Після цього необхідно записати дані в систему, так щоб запит був оброблений вузлом з ідентифікатором 0. Цього можна досягнути переглянувши лог, в якому вказано актуальні ідентифікатори вузлів, та направивши запит на запис на відповідний сервер. Після цього слід виконати запит на будь-який вузол, та переконатись, що відповідь на нього співпадає з очікуваним результатом.

3.4 Висновок по розділу

В даному розділі було розглянуто загальні вимоги до тестування розподіленого сховища даних та проаналізовано оптимальні методи тестування виходячи з функціональних вимог до програмного продукту та умов його експлуатації.

Було вирішено приділити особливу увагу навантажувальному тестуванню, так як воно не тільки дозволить провести перевірку швидкодії сховища, а й дозволить виявити неполадки в механізмах реплікації та фрагментації даних, а також при вилученні та додаванні вузлів. Ці типи помилок можуть з’являтись лише при великому навантаженні на систему, тому таке тестування є необхідним.

Також було розглянуто приклад тесту з перевіркою функціональності обробки запитів зчитування даних. Інші тести розглянуті в додатку В “Програма та методика тестування”.

Після проведення всіх необхідних тестів програмне забезпечення може бути введене в експлуатацію.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Програмне забезпечення працює на однорідному кластері з обчислювальних вузлів, тому для його розгортання необхідно, щоб між усіма вузлами було стійке мережеве з’єднання. На вузлах повинен бути встановлений Java Runtime Environment восьмої версії, або новіший.

Детальні інструкції по розгортанню програмного забезпечення вказані в додатку Г “Керівництво системного програміста”.

4.2 Робота з програмним забезпеченням

Так, як дане програмне забезпечення призначення, для використання в якості сховища даних для інших програмних продуктів, робота з ним описана в додатку Д “Керівництво програміста”.

4.3 Висновок по розділу

В цьому розділі було розглянуто розгортання та роботу з програмним продуктом. Оскільки даний продукт призначений для використання в якості сховища даних у інших програмних продуктах, його користувачами є системні адміністратори та програмісти.

В додатку Г “Керівництво системного програміста” було детально описано процес запуску сховища, приєднання та вилучення вузлів з нього, а також описано роботу системи моніторингу.

В додатку Д “Керівництво програміста” було розглянуто API продукту та важливі для програміста, як для користувача даної системи, особливості його роботи.

ВИСНОВКИ

Під час виконання даного дипломного проекту було проаналізовано наявні на сьогодні технічні рішення в області розподіленого зберігання даних, після чого було виділено ряд недоліків таких систем та розглянуто способи їх вдосконалення. Щоб досягнути цього було проведено аналіз предметної області, визначено основні варіанти використання розподілених сховищ даних та сформовано ряд вимог до розроблюваного програмного забезпечення.

В результаті дослідження розроблено розподілене сховище даних, що має підтримку динамічного горизонтального масштабування, що практично не впливає на його швидкодію, може працювати в умовах розділу мережі та виходу з ладу вузлів, а також запобігає втраті даних з допомогою вирішення конфліктів у даних на клієнтській частині.

Для досягнення цих властивостей було використано поєднання таких алгоритмів як узгоджене хешування та векторний годинник для реплікації та фрагментації даних у системі. Оскільки система є розподіленою, для поширення інформації в ній використано механізми gossip протоколу та двох коркової фіксації, реалізовані на базі акторної моделі.

Всі взаємодії в розробленій системі є асинхронними та часто відбуваються між різними вузлами системи, тому їх було детально описано та проілюстровано з допомогою діаграм.

Так, як розроблене програмне забезпечення застосовується як сховище даних для інших програмних продуктів, особливу увагу було приділено опису процесу його розгортування та експлуатації системними адміністраторами та програмістами, а також реалізовано та описано систему його моніторингу та логування. Крім цього було проведено етап тестування та описано його процес.

Розроблене розподілене сховище даних може бути застосоване при реалізації високонавантажених систем, та систем що мають вимоги до високої надійності.

ПЕРЕЛІК ПОСИЛАНЬ

1. Dynamo: Amazon's Highly Available Key-Value Store / G. DeCandia, D. Hastorun, M. Jampani. – ACM Symposium on Operating Systems Principles, Stevenson, 2007 – 16p.

2. Consistent Hashing with Bounded Loads / V. Mirrokni, M. Thorup, M. Zadimoghaddam. – Google Research, New York, 2016 – 37p.

3. Epidemic Algorithms for Replicated Database Maintenance / A. Demers, D. Greene, C. Hauser. – ACM Symposium on Principles of Distributed Computing, 1987 – 28p.

4. Timestamps in Message-Passing Systems That Preserve the Partial Ordering / Colin J. Fidge – Australian Computer Science Conference, 1988 – 11p.

5. CRDTs: Consistency without concurrency control / Mihai Letia, Nuno Preguiça, 2009 – 17p.

6. Розподілені бази даних [Електронний ресурс]: (Стаття) / СумДУ. – Електрон. дан. (1 файл). – 2019. – Режим доступу: https://elearning.sumdu.edu.ua/free_content/lectured:89b3d175c06a6b137e410cb14821d0e94549ad5a/latest/44605/index.html. – Назва з екрана.

7. Розподілена база даних [Електронний ресурс]: (Стаття) / Wikizero. – Електрон. дан. (1 файл). – 2019. – Режим доступу: https://www.wikizero.com/uk/%D0%A0%D0%BE%D0%B7%D0%BF%D0%BE%D0%B4%D1%96%D0%BB%D0%B5%D0%BD%D0%B0_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85. – Назва з екрана.

8. Теорема CAP [Електронний ресурс]: (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2019. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0_CAP. – Назва з екрана

9. C. Horstmann, Scala for the Impatient – pub. Addison-Wesley Professional, San Francisco, 2012 – 412p.

10. Documentation | Akka [Електронний ресурс]: (Стаття) / Akka. – Електрон. дан. (1 файл). – 2019. – Режим доступу: https://akka.io/docs/. – Назва з екрана.

11. CRDT: Conflict-free Replicated Data Types [Електронний ресурс]: (Стаття) / Habr. – Електрон. дан. (1 файл). – 2019. – Режим доступу: https://habr.com/ru/post/418897/. – Назва з екрана.